PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04Q — A2

(11) International Publication Number: WO 98/32289

(43) International Publication Date: 23 July 1998 (23.07.98)

(21) International Application Number: PCT/US98/00771

(22) International Filing Date: 16 January 1998 (16.01.98)

(30) Priority Data:
60/035,623 17 January 1997 (17.01.97) US

(71) Applicant: THE BOARD OF REGENTS OF THE UNIVERSITY OF WASHINGTON [US/US]; Office of Technology Transfer, 1107 N.E. 45th Street, Seattle, WA 98105 (US).

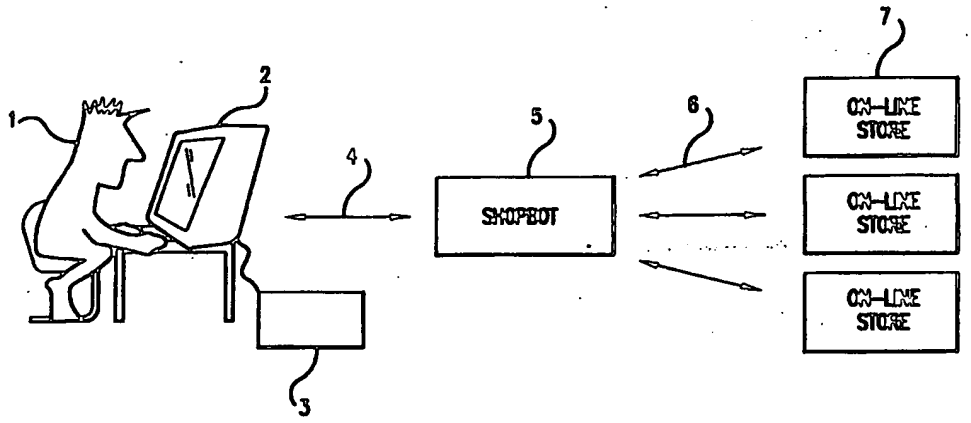
(72) Inventors: DOORENBOS, Robert, B.; 1154 N.W. 59th Street #A32, Seattle, WA 98107 (US). ETZIONI, Oren; 5820 57th Avenue N.E., Seattle, WA 98105 (US). WELD, Daniel, S.; 4315 N.E. 43rd Street, Seattle, WA 98105 (US).

(74) Agents: MORRIS, Francis, E. et al.; Pennie & Edmonds LLP, 1155 Avenue of the Americas, New York, NY 10036 (US).

(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
*Without international search report and to be republished upon receipt of that report.*

(54) Title: METHOD AND APPARATUS FOR ACCESSING ON–LINE STORES




(57) Abstract

This invention provides a computer–implemented agent that assists a user in accessing network linked on–line stores. In one aspect, the invention is a method for intelligently routing a user query to on–line stores relevant to that query, extracting relevant data fields from received responses, and intelligently presenting the extracted data in order of estimated interest. In another aspect, the system of this invention implements one or more steps of the method in a centralized or distributed manner on one or more network linked computers. Further, this invention provides a novel heuristically guided process by which the agent is capable of automatically acquiring sufficient information on the characteristics of on–line stores for it to access and shop at those stores.

***FOR THE PURPOSES OF INFORMATION ONLY***

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AL | Albania | ES | Spain | LS | Lesotho | SI | Slovenia |
| AM | Armenia | FI | Finland | LT | Lithuania | SK | Slovakia |
| AT | Austria | FR | France | LU | Luxembourg | SN | Senegal |
| AU | Australia | GA | Gabon | LV | Latvia | SZ | Swaziland |
| AZ | Azerbaijan | GB | United Kingdom | MC | Monaco | TD | Chad |
| BA | Bosnia and Herzegovina | GE | Georgia | MD | Republic of Moldova | TG | Togo |
| BB | Barbados | GH | Ghana | MG | Madagascar | TJ | Tajikistan |
| BE | Belgium | GN | Guinea | MK | The former Yugoslav Republic of Macedonia | TM | Turkmenistan |
| BF | Burkina Faso | GR | Greece | | | TR | Turkey |
| BG | Bulgaria | HU | Hungary | ML | Mali | TT | Trinidad and Tobago |
| BJ | Benin | IE | Ireland | MN | Mongolia | UA | Ukraine |
| BR | Brazil | IL | Israel | MR | Mauritania | UG | Uganda |
| BY | Belarus | IS | Iceland | MW | Malawi | US | United States of America |
| CA | Canada | IT | Italy | MX | Mexico | UZ | Uzbekistan |
| CF | Central African Republic | JP | Japan | NE | Niger | VN | Viet Nam |
| CG | Congo | KE | Kenya | NL | Netherlands | YU | Yugoslavia |
| CH | Switzerland | KG | Kyrgyzstan | NO | Norway | ZW | Zimbabwe |
| CI | Côte d'Ivoire | KP | Democratic People's Republic of Korea | NZ | New Zealand | | |
| CM | Cameroon | | | PL | Poland | | |
| CN | China | KR | Republic of Korea | PT | Portugal | | |
| CU | Cuba | KZ | Kazakstan | R | Romania | | |
| CZ | Czech Republic | LC | Saint Lucia | RU | Russian Federation | | |
| DE | Germany | LI | Liechtenstein | SD | Sudan | | |
| DK | Denmark | LK | Sri Lanka | SE | Sweden | | |
| EE | Estonia | LR | Liberia | SG | Singapore | | |

# METHOD AND APPARATUS FOR ACCESSING ON-LINE STORES

## 1. FIELD F THE INVENTION

The field of this invention relates to information access over networks, and specifically to providing assistance in accessing on-line electronic stores by automatically retrieving product descriptions in response to a user product query.

## 2. BACKGROUND

The exponential growth of private intranets and the public Internet has produced a daunting labyrinth of increasingly numerous on-line electronic stores and product information databases. Almost any type of product is now available somewhere, but most users cannot find what they seek, and even expert users waste copious time and effort searching for appropriate on-line stores or other product information sources. One problem is simply the increasingly large number of available sources that are beyond the comprehension of a single user. A second problem, along with this growth in available on-line stores and product information, is a commensurate growth in software utilities and methods to manage, access, and present this information. Each utility has a different and often unique interface and set of commands and capabilities, and is appropriate for a different set of users and a different set of information types and sources. Thus sheer diversity of available utilities creates problem for users comparable to that created by information explosion. Users are now faced with the twin problems of which tool to use to inquire at which information source.

In the past efforts have been made to provide users with automatic, computer assisted services that can help solve these twin problems of the network revolution. For example, AI researchers have created several prototype software agents that help users with e-mail and netnews filtering (Patti

Maes et al., 1993, Learning interface agents, *Proceedings of AAAI-93*), agents that assist with World Wide Web browsing (H. Lieberman, 1995, Letizia: An agent that assists web browsing, *Proc. 15th Int. Joint Conf. on A.I.* pp. 924-929; Robert Armstrong et al., 1992, Webwatcher: A learning apprentice for the world wide web, *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pp. 6-12, Stanford University, AAAI Press), agents that schedule meetings (Lisa Dent et al., 1992, A personal learning apprentice, *Proc. 10th Nat. Conf. on A.I.*, pp. 96-103; Pattie Maes, 1994, Agents that reduce work and information overload, *Comm. of the ACM* 37(7):31-40, 146; Tom Mitchell et al., 1994, Experience with a learning personal assistant, *Comm. of the ACM* 37(7):81-91), and agents that perform internet-related tasks (O. Etzioni et al., 1994, A softbot-based interface to the internet, *CACM* 37(7):72-75). Increasingly, the information such agents need to access is available on the World Wide Web. Unfortunately, even a domain as standardized as the WWW has turned out to pose significant problems for automatic software agents. For one, although Web pages are universally written in Hypertext Markup Language ("HTML"), this language merely defines the format of information display, making no attempt to hint at its meaning or semantic content. Currently, no accepted "semantic markup language" for the Web exits, nor is one likely to adopted universally. The Internet can be expected to pose even greater problems.

Thus, the advent of intranets, the Internet, and the World Wide Web have posed several fundamental problems for the automatic services or agents designed to assist users to find relevant information. First, no one such service has heretofore provided sufficient additional value to replace the use of a Web browser having access to existing on-line stores and directories or indices such as Yahoo or Lycos. Second, such services have not yet been able to understand and competently parse relevant product information from the responses returned from a wide variety of Internet and Web

on-line electronic stores. Third, existing services and agents have not been easy to adapt to the ever-increasing numbers of stores with their ever-changing response formats. This is due to the individualized, hand-coded interface to each Internet service and Web site utilized by existing agents (Yigal Arens et al., 1993, Retrieving and integrating data from multiple information sources, *International Journal on Intelligent and Cooperative Information Systems* 2(2):127-158; O. Etzioni et al., 1994, A softbot-based interface to the internet, *CACM* 37(7):72-75; B. Krulwich, 1995, Bargain finder agent prototype, Technical report, Anderson Consulting; Alon Y. Levy et al., 1995, Data model and query evaluation in global information systems, *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval* 5(2); Mike Perkowitz et al., 1995, Category translation: Learning to understand information on the internet, *Proc. 15th Int. Joint Conf. on A.I.*). Preferably, a service or agent should be able to access a new or changed Internet on-line store in order to automatically learn how to retrieve relevant information from the source.

## 3. SUMMARY OF THE INVENTION

It is a broad object of this invention to solve these fundamental problems by a method and system that provide a personalized network shopping robot, called a "shopbot." A shopbot acts as a user's intelligent assistant by tracking available network product sources or on-line stores, knowing the relevant information and features of each particular product source, and upon user request determining which product sources are relevant to a given query, forwarding the query to the most relevant product sources, understanding the responses returned from each source, and integrating and intelligently presenting the query results to the user.

The shopbots of this invention possess several advantages, including the following. First, a shopbot returns only the relevant product information to the user.

On the one hand, each user query is forwarded only to the on-line stores in the product domain of interest to a user. On the other hand, responses returned from on-line stores are parsed and understood so that only the relevant product data items are extracted for user presentation. Duplicate, stale, irrelevant, and mere formatting information items are discarded. Second, a shopbot is fast. Since it automatically searches the relevant on-line stores in parallel, it can present product information as quickly as the fastest primary source returns a response. Despite changing conditions which cause different information sources to fluctuate in speed, a shopbot remains as fast as the fastest on-line store. Stores that have no information to return to a query do not slow the user since the shopbot simply ignores them. Third, shopbots are easily adapted to the ever-increasing number of on-line electronic stores with ever-changing response formats. Shopbots utilize a novel and simple method for describing information sources. A source description is a short and easily understandable collection of strings.

Therefore, in one aspect the invention includes a method for efficient access to product information sources on a network comprising preferably one or more of the following steps: receiving a user query for product information; determining the product information sources or on-line stores in the correct product domain relevant to this query; retrieving a description of each information source; formatting a form by which to access these on-line stores according to the query and to the retrieved description in a manner suitable for each product information source; transmitting the form to the on-line store; receiving responses from the product information sources; for each on-line store, understanding and extracting the relevant data fields according to the retrieved description; and presenting to the user the relevant data from each on-line store in an intelligent manner ranked by an estimate of its interest to the user. Advantageously, these steps are performed in

parallel to the greatest extent possible. In particular, at least, all queries are transmitted to all relevant information sources in parallel without waiting for intervening responses.

In another aspect, this invention includes a heuristically guided process by which a shopbot can determine automatically the description of an on-line store. The heuristics are collected into domain descriptions for each product domain to be accessed. The domain description include rules defining typical attributes of products in this domain and seed knowledge to generate training examples from an on-line store This process includes one or more of the following steps: searching for likely product query forms at a product information source or on-line store; for each likely query form querying the on-line store both with products not likely to be carried by the store and also with popular products likely to be carried; and selecting the form for future queries which results in the greatest query success.

In a further aspect the invention comprises a computer system and apparatus for performing one or more steps of the method of this invention. The user has a presentation device attached to a network to which is also attached a plurality of product information sources or on-line stores. The presentation device receives user queries and displays shopbot responses. Further, the presentation device performs one or more of the steps of the method of this invention. One or more of those steps not performed on this device can advantageously be performed on network attached shopbot server computers, which respond to functional requests from the user device. Optionally, the user device can range from a diskless hand-held terminal, to a PC, to a work station, and so forth.

## 4. BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects, and advantages of the present invention will become better understood by reference to the accompanying drawings, following description, and appended claims, where:

Fig. 1 illustrates generally a shopbot of this invention;

Fig. 2 illustrates an exemplary user interface of embodiments of the shopbot of Fig. 1;

Fig. 3 illustrates exemplary functional components of the shopbot of Fig. 1;

Fig. 4 illustrates alternative hardware embodiments of the shopbot of Fig. 1;

Fig. 5 illustrates an exemplary product query page for an on-line store;

Fig. 6 illustrates an exemplary product query response page for an on-line store;

Fig. 7 illustrates in general the learning phase of a shopbot of Fig. 1;

Fig. 8 illustrates in more detail the learning phase of a shopbot of Fig. 1; and

Fig. 9 illustrates in more detail the shopping phase of a shopbot of Fig. 1.

## 5. DETAILED DESCRIPTION

For clarity of disclosure, and not by way of limitation, the detailed description of a shopbot of this invention is presented as a method for accessing on-line stores and as a system or apparatus implemented to perform that method.

In the following, first an overview of the invention is presented followed, second, by a detailed discussion of individual components.

## 5.1. OVERVIEW OF SHOPBOT ARCHITECTURE

A shopbot method or system of this invention comprises software and hardware facilities that function together in one or more network attached computers to assist a user to access product information stored in network attached servers (known herein alternatively as "on-line stores," "stores," or "vendors"). Fig. 1 generally illustrates the relationships of a shopbot to a user and to networked on-line stores or product information sources. For example, user 1 accesses user computer 3 through standard interface devices, such as monitor 2. In the course of work, the user needs information from on-line stores 7, attached to the user computer through various network links, such as network links 4 and 6. Since the on-line stores are many, the user can benefit from assistance in finding needed comparative product information from relevant on-line stores. This assistance is provided by shopbot 5, which maintains an awareness of available on-line stores and their characteristics, and queries them through links 6 on behalf of, or as an agent of, the user. Alternatively, shopbot 5 can partly or wholly reside on user computer 3 or be partially or wholly distributed on the network and accessed by the user through link 4.

Groups of on-line stores 7 selling similar sorts of products are grouped into conceptual classes called product domains. For example, one domain can be that of electronic stores for pop/rock music CDs, and another can be that of electronic stores for computer software or hardware products.

In a preferred embodiment illustrated in Fig. 3, shopbot is composed of three major functional modules: user interface 36, integrator 37, and I/O manager 41. Briefly, the user interface module interacts with the user to receive user queries for information, and to format and present information responses received from the network attached on-line stores. Advantageously, the user interface is adapted to the specific product domain being accessed. In a shopping phase, the integrator module accepts a user product query from the user interface module, formats it for network

transmission to each on-line store of the product domain, receives product responses from these stores, understands thes responses, and passes the relevant portions of the responses back to the user interface module for display to the user. In a learning phase, the integrator module is capable of accessing a new or changed on-line store and querying it in the process of determining a store description for use during the shopping phase. The I/O manager module performs hardware, operating system, and network specific interfacing for the user interface and integrator modules so that porting a shopbot to different hardware platforms, operating systems, or networks requires only limited changes in well-modularized code.

In particular alternative implementations, either or both of the user interface or the I/O manager modules may be absent. For example, the functions of these modules can be already performed by other operating system components. A shopbot can provide only one or more of the facilities disclosed without providing others. For example, in some embodiments, a shopbot can simply format queries and understand responses, with the on-line store description being externally supplied. In these cases, a learning shopbot can be present elsewhere on the network and can provide vendor description to other shopbots on request. Finally, as is known to those of skill in the art, the functions performed by the described modules may be divided or grouped in alternative fashions among a greater or lesser number of modules.

In the absence of specific preferences, the processes of this invention can be implemented in a procedural programming language, such as C, or an object oriented programming language, such as C++, on the disclosed hardware configurations.

The User Interface Module

In more detail, the user interface module has both important functionality that is common to shopbot user interfaces, whatever the product domain to which shopbots are directed, and also has adaptations to the particular product domain of a particular shopbot. Turning first to the preferable common functions, one such is the ability to remember a user's preferences for interacting with a shopbot. Such remembered preferences include, for example, screen display format including preferred product attribute fields and preferred result sort order, the number or identity of on-line stores to query, and so forth.

The user interface module preferably provides one or more windows on the user's screen with several defined views of the query satisfaction process along with certain common user controls, such as screen buttons, for manipulating these windows. In one window, the user interface module presents lists of the on-line stores being consulted with each source symbolically represented as, for example, a network address, an icon, or another compact screen representation. Also displayed is a count of the total number of unique product items received currently. Optionally, clicking on the screen representation of an on-line store opens a further window with either information about this on-line store, or a display of the responses received from it, or access to the on-line store over the network, etc.

In addition to such common functions and controls, a shopbot user interface module preferably implements specific designs, formatting, and fields suitable to the information domain for which it is designed. For example, a shopbot for comparison shopping in a product domain of software stores can have a particular interface presentation containing labeled fields for product name, model, hardware requirements, operating system requirements, price, and so forth. A shopbot for a product domain of pop/rock CD stores can have a particular interface presentation containing

labeled fields for artist, group, data, publisher, price, and so forth.

In one shopbot embodiment, most functions and modules reside on network attached servers which a user accesses remotely. For example, the user may access a shopbot over the Internet with the World Wide Web protocols utilizing a web browser, such as Netscape. In this case the user interface builds HTML formatted pages which are transmitted over the network by the I/O manager. Fig. 2 generally illustrates the user display from an example of such an embodiment, which is further directed to the information domain of on-line, electronic software stores. The shopbot display of Fig. 2 is divided into three sections. Section 11 is a title section generally indicating that this display has results from a shopbot. A shopbot preferably also has a specific input query screen. Section 12 presents the list of on-line stores currently being consulted represented by their WWW addresses 15, which are selectable to provide further information or direct WWW access. At 16 in section 12, those sources which have already returned query results are similarly represented. Section 13 presents the results received so far formatted in accordance with this particular product domain into sections for the major PC operating systems. Each individual item returned, for example item 16, is formatted with product name, price, and an address for the originating on-line store. In this implementation, information display is controlled with the window scrolling and control facilities built into the web browser. This user interface is implemented as a HTML formatted page created at a shopbot server and transmitted to the web browser.

In another embodiment, the functions and modules can reside on the user's local computer. In this case, the shopbot sends queries, receives responses, and formats results locally. The I/O manager utilizes the facilities of the local operating system for user interaction.

Although the user interface is described primarily in terms of windows and buttons, one of skill in the art will

recognize that this invention is adaptable to other display paradigms that provide for display of information and input of user commands. For example, the user interface module can control the entire screen and present graphical displays without intervention of a windowing system.

The user interface module is preferentially implemented with an object oriented programming language supplemented with a class library providing windowing functions. A preferable implementation uses the Java language together with the java.awt package. See, for example, Flanagan, 1996, Java In A Nutshell, O'Reilly & Associates, sections 5 and 19.

**The Integrator Module**

Fig. 3 illustrates the preferred functional modules, data bases, and functional interrelationship both of shopbot 30 in general and of integrator module 37 in particular. The integrator preferably consists of three functions: learning phase modules 39, database 40 of product domain and on-line store descriptions, and shopping phase modules 38. These components are introduced here and described in detail in the following.

In a shopping phase, a comparison shopping query 31 is delivered to the integrator by means of the user interface module 34. The integrator calls the shopping phase modules 38 to chose the on-line stores appropriate to the product domain of the query. Next the integrator retrieves the store descriptions for these stores from database 40. Alternately, each shopbot can be specialized to only one product domain, in which case all the store descriptions are retrieved from database 40. In this case, if these are only a small number of stores the store descriptions can alternately be stored in a table in memory. These descriptions of the on-line store and its requirements comprise a set of strings, to be subsequently described in detail, are used by shopping phase modules 38. These modules, first, retrieve product query pages from each on-line store, second, format the user query

into fields on each of the pages, and third, then submits the filled-in pages to each store in parallel.

When stores 33 return responses, shopping phase modules 38, using strings in the store description, extract data from the responses and place it into a list of data fields, called a tuple format, relevant to the particular product domain. Optionally, each tuple can be assigned a priority order using a method appropriate to the particular user query. Finally when screen display manager of user interface 36 requests data to present to the user, perhaps in response to a more-button request, the shopping phase modules pass the tuples to user interface module 34, sorted in priority order if a priority is determined. For example, if the product domain relates to on-line software stores, then the tuples optionally contain such relevant fields as product name, manufacturer, software version number, operating system required, price, etc. An exemplary priority order of the tuples can be by price, by delivery delay, or other factor at user preference. The user display is controlled according to stored user preferences 35.

In a learning phase, the location of a new or changed on-line store is delivered to the integrator. The product domain is externally supplied to the integrator along with the on-line store identification. Alternately, the integrator can call learning phase modules 39 to determine the product domain of the identified store. Next the integrator retrieves the product domain descriptions for the domain from database 40. This domain description includes heuristic rules to be subsequently described which guide the learning phase modules in automatically acquiring store descriptions. Next, the learning phase modules interact with on-line store 33 in a manner tailored by the heuristic rules from the domain description in order to determine the strings of a vendor description for this on-line store. When a successful vendor description has been determined, it is stored in database 40 for use by the comparison shopping phase modules.

The I/O Manager Module

The I/O manager module 41 of Fig. 3 performs hardware, operating system, and network specific interfacing for the integrator module. Network interfacing includes the tasks of sending requests and receiving responses from network linked on-line electronic stores according to protocols recognized by the stores. Since a preferred application of the shopbots of this invention is to shopping on Internet, the I/O manager is responsible for implementing the relevant protocols of the WWW, including TCP/IP, HTTP, and so forth. Optionally, I/O manager 41 can temporarily cache pages and other data in order to improve response time. Operating system interfacing includes the task of window management for the user interface module and access to the database services, if present.

Preferably, the I/O manager is constructed from commercially available protocol stacks, windowing libraries, such as the Java.awt package, and other tools. In some implementations, more or less of the I/O manager functions can be performed by other system components on the network attached computer. Optionally, the I/O manager is designed to be scalable to multiple machines, to not require multi-threaded or reentrant code, and to be cross platform and persistent.

The Shopbot System

The preferred functional structure of a shopbot can be assigned to system hardware components in various alternatives. The preferred alternative in any case depends on which allocation of function achieves a rapid response and reasonable cost. Fig. 4 generally illustrates exemplary shopbot hardware embodiments and options in view of the previous general description. It illustrates the interrelationship of user computer elements 51-56, network 57, on-line stores 58, and shopbot server computers 59-61. Computer 51 is a user computer including a processor, memory, and various attached peripherals. Such peripherals include display device 52, or other device for user interaction,

network attachment 54, optional hard disk storage 53, and so forth. Computer 51 can be alternatively a network device without permanent storage, a PC, a work station, or more powerful computer. It is preferred that computer 51 be a PC or a work station running one of the Windows operating systems, the Macintosh operating system, or UNIX. Present in the memory of user computer 51 is, among other software, local shopbot software 55 and local system components 56. The local shopbot software implements one or more of the shopbot functions. The local system components can include, for example, a web browser.

Network 57 can be any network with a plurality of attached on-line stores 58, which can be optionally conceptually classified by type of products sold into a plurality of product domains. In a preferred embodiment, network 57 is the public Internet or a private intranet supporting the TCP/IP suite of protocols, including such user level protocols as FTP, HTTP, and so forth. The on-line stores are server computers which make their stored product information available using the protocols supported by network 57. Such information can include product type, model, and manufacturers and store price and availability.

In such a network, a shopbot can have various embodiments. In an entirely local embodiment, all shopbot functions reside in local shopbot software 55 on user computer 51, which in this embodiment must have sufficient processing and storage capabilities. In alternative embodiments, one or more of the disclosed shopbot functions can be distributed on other network attached computers.

For example, computer 59 is a store/domain description server for accepting requests for downloading on-line store description or product domain descriptions stored in its database. This database can be stored in memory or on disk using any data management system capable of storing and retrieving compact textual descriptions. Computer 60 is a learning-phase server for performing the computationally more intensive tasks of determining new on-line store descriptions

and providing these description either to database computer 60 or directly to local shopbot 55. Computer 61 is a shopbot server for performing the shopping modules function by accepting user queries and returning search results, perhaps using the facilities of store/domain description server 59 or learning phase server 60. With these network servers, local shopbot software preferably only supports the user interface, which may be performed entirely by a web browser. Alternately, it can further include the shopping phase modules, which make query routing requests to query server 59 and store description requests to store description server 60. Further, it can include one or both of these latter functions.

The various computers of a shopbot system can be provided with software for performing the methods of this invention either from computer readable media or by loading across a network. This invention is adaptable to known magnetic and optic media, such as disks, tapes and CD-ROM.

## 5.2. COMPARISON SHOPPING AT ON-LINE STORES

This section describes the regularities of on-line stores that are advantageous to shopbots, and further describes the particular comparison shopping task at which shopbots provide assistance. On-line stores available according to the WWW protocol send HTML formatted documents to a user in order to announce the store, display its products, and receive orders. For a general description of the HTML document description language, see, e.g., ____. These store typically are characterized by several regularities in the presentation of their HTML documents. First, their presentation is designed so that customers can find available products and product information quickly. Often, on-line stores provide simple methods to move quickly from the store's home page, that is the document first accessed when a customer visits the store, to a form which a customer can fill out in order to request product information. Fig. 5 illustrates exemplary product

information request form 500 adapted to an wide selection of products. By filling in some or all of the available search fields, a customer can search for products. For example, the customer can search for products of a certain category having certain words in their description by selecting a category from field 501 and entering search words in field 501. The customer submits the form to the on-line store by clicking box 503. Line 504 presents "navigation" aids to help the customer access other HTML information documents from the store.

Submission of such a search form causes server computers at the store to search a database of product information describing the on-line store and then return product information to the user, also formatted as one or more HTML documents. On-line stores attempt to create a sense of distinctive identity by using a uniform look and feel for their documents. Although stores differ widely in their product description formats, a particular store advantageously describes all available products in a consistent format. In particular, while different stores use different product description formats, substantially all use vertical separation and white space, that is blank areas of a document, to facilitate customer comprehension. For example, stores start each product descriptions on a separate logical line. Fig. 6 illustrates exemplary product description form resulting from a query using the words "iomega jaz." Here, each product with those words in its description is presented on a separate line, such as lines 506. Line 507 is exemplary header information, and line 508 is exemplary trailer information.

On-line vendors respect such regularities because they facilitate comprehension and, thus, sales to human customers. However, these regularities are exploited by a shopbot. Presence of a search form allows a shopbot to simply find product descriptions in a store-independent manner. Regularities in the resulting product description forms permit a shopbot to learn how to access such a store

substantially independently of operator assistance. In particular, these regularities allow the learning procedure to incorporate a strong bias, and thus require only a small number of training examples. In other implementation, a shopbot can provide assistance at on-line stores lacking such forms and regularities. However, for such stores substantial operator assistance can be required in order to tailor a shopbot for such a store.

A preferred implementation of shopbots assists users in comparison shopping. In comparison shopping, a customer seeks the on-line store from which to purchase a particular product that is most advantageous according to some criteria. Therefore, comparison shopping identifies a group of on-line stores that sell the particular product desired by a user, and then ranks the stores in the group based on the customer criteria, *e.g.*, price, speed of delivery, and so forth. For example, in the domain of computer-products stores, a comparison-shopping shopbot can help answer: "find the lowest price for the Macintosh version of Abode Photoshop." In general, a shopbot functions according to the following method. Upon receiving such a request, it determines the relevant stores and accesses them in parallel. Next, it searches for the indicated product by retrieving, filling out, and submitting the HTML product search forms available at each on-line store. In a preferred embodiment, parsing and filling out these search forms is done according to a vendor description, which comprise sets of recognition strings. The stores return to the shopbot HTML pages describing their terms for the indicated product. The shopbot parses these returned pages, again preferably, according to the vendor or on-line store descriptions strings. The parsing procedure ignores any header and trailer fields and parses the remaining HTML formatting code into logical lines matching a learned product description format. Returned pages matching a failure template that the shopbot has learned indicates that the search failed at this on-line store are discarded. Finally, the shopbot sorts the

product information, e.g., by ascending order of pric , and generates a summary for the user.

In more detail, the total comparison-shopping problem is solved in two phases. In a first learning phase described subsequently, a shopbot analyzes on-line stores to learn an on-line store description, for handling HTML pages from the site. This phase is more computationally expensive, but is performed in advance of actual customer comparison shopping, and needs to be done only once per store. However, if a vendor "remodels" the store with different HTML formatting, providing different search forms or different product description page formats, then this first learning phase is repeated for that vendor. In a second shopping phase, which is less computationally expensive, the learned information is actually used by a customer for comparison shopping. The shopping phase is implemented according to the previously described shopbot architecture for rapid parallel access of relevant on-line stores.

In particular, a shopbot's implementation and graphical user interface advantageously utilizes certain important principles. First, the shopbot in the comparison shopping phase is fast. Because most of the computational work has been in advance of shopping in the learning phase, the shopping phase can be fast. In fact, although the most time consuming step is fetching pages over the network, since such fetching is done in parallel across all vendors, a shopbot is faster than an expert human. Second, a shopbot provides its user with continual feedback informing the user which vendors are being contacted and what prices have been found so far and permitting user interrupts at any time. Third, the shopbot provides the user with enough context around any information it extracts so that the user can verify its conclusion or investigate manually. For the user's convenience, ShopBot indicates the store's home page, the search form it used, and each full product description found.

In summary, Table 1 outlines the input and output to a comparison shopping task.

TABLE 1 - COMPARISON SHOPPING

| |
|---|
| Given: |
| 1. A description on the particular on-line shopping domain, including information about product attributes, *e.g.*, name, manufacturer, price, and so forth, useful for discriminating between different products and between variants of the same product and typical popular products; |
| 2. URL's for the home pages of possible on-line stores; |
| 3. An attribute *A*, *e.g.*, the price, by which the user wants to compare vendors; and |
| 4. A specification of the desired product in terms of the values of selected attributes, *e.g.*, name is Photoshop. |
| Determine: |
| 5. The set of on-line stores where the desired product is available, sorted by *A*. |

Generally, input to the learning phase is items 1 and 2. Using this information and a constrained learning process, a shopbot is capable of learning on-line store descriptions for accessing product information in the on-line stores. Items 3 and 4 are input to the shopping phase. A user inputs attributes of the products of interest in a manner consistent with the domain, and the shopbot retrieves product information from the relevant on-line stores.

Although, this description of a shopbot is directed to the comparison shopping task, it will be apparent to those of skill in the art that a shopbot can equally well be constructed for other general shopping tasks involving network search. Further, for tasks in which the relevant on-line stores present significant regularities, a learning phase can be constructed which learns the relevant on-line store descriptions.

### 5.3. LEARNING ON-LINE STORE DESCRIPTIONS

This subsection describes the first learning phase of shopbot processing in which a shopbot learns the information necessary for it to assist a user in the second comparison-shopping phase which can be repeatedly performed as long as

the learning process preferably proceeds with a minimum of training examples and in an unsupervised manner. To minimize disruption of an on-line store's activities, it is preferable that shopbot learning retrieve a strictly limited number of HTML documents from the store. Further, it is preferable that operator intervention be minimized or eliminated in order to a shopbot to be able to comparison-shop at a new or revised on-line store. In this subsection, first, the processing, input and output of the learning process are described in general, followed, second, by details of each major step.

In general, a shopbot can learn a vendor description for an on-line store by using heuristics which exploit the regularities typically present in an on-lines store's HTML documents. These heuristics strongly direct the learning process according to these regularities. First, these heuristics assume that every product description is somehow vertical-space-delimited, by, e.g., starting a paragraph, a new row in a table, a new line, and so forth. Such vertical-space-delimitation is specified by HTML tags such as <p>, <tr>, <li>, or <br>. Accordingly, after removing header and trailer information, the heuristics divide remaining HTML code of each page into "logical lines" representing groups of vertical-space-delimited text. Second, the heuristics assume that every product is described in the same format. Accordingly, each "logical line" that is found is abstracted into a "line description" by removing the arguments from HTML tags and replacing all occurrences of intervening text with the variable "text." The most successful such line description is used to describe that on-line store's product description pages. In fact this last regularity is expected since most on-line stores retrieve product information from a relational database with a program to create a custom information page in a simple format.

In more detail, the learning process is product domain independent. All domain dependence is input to the learning process as data contained in a domain description.

Therefore, in order to shop in a new product domain whose on-line stores share described regularities, the learning modules merely need to fetch the appropriate domain description from storage. A domain description contains three categories of information: a description of the product attributes, heuristics for understanding on-line store pages, and seed knowledge to bootstrap learning. Product attributes are categories relevant for describing products in this domain. For example, for a computer software domain, product attributes can include product name, manufacturer, price, hardware requirements, operating system requirements, and so forth. Heuristics for understanding on-line store pages recognize the terminology used in the search. These heuristics are preferably of the form of rules whose antecedents match typical words used to describe input fields on query forms and whose consequents specify which product query attributes to fill into the input fields. For example, turning to Fig. 5, for search forms such heuristics can recognize the words "product category" in field 501 or "description" in field 502 as possibly identifying input fields for product attributes or description, respectively. Finally, sample seed knowledge is used as test queries to begin learning of on-line store product description pages. This knowledge includes products almost certainly not in the domain in order to learn to recognize search failure pages. It also includes common products likely to be present in many stores in order to learn successful product description pages. For example, for computer software seed knowledge can include the products "Microsoft Encarta," "Abode Photoshop".

In addition, although this detailed description of the learning phase modules of this invention is directed toward their application toward learning descriptions of on-line stores, the methods of these modules are not so limited. They are equally applicable to any information source that obeys the previously mentioned format regularities. In particular, they are applicable to learning how to access any information source that has an information search form, that

returns failure pages upon information search failure, that returns information pages upon search success having a uniform format in which information items are separated by recognizable formatting codes, such as HTML codes for vertical white space.

Summary Of The Learning Process

On-line stores typically provide a search form that a user can fill in with, *e.g.*, the name and manufacturer of a desired product and then submit, *e.g.*, by clicking on a "submit" or "search" button. The on-line store responds by returning a page containing product information in a consistent HTML format for user scrutiny. Thus, for comparison-shopping, a shopbot accesses the page containing the search form, properly fills in the accessed search form, and then extracts relevant product information from the returned page(s). For the latter step, the format of information on a product information page needs to be represented.

Therefore, the shopbot learning problem is comprised of three sub-problems. A first sub-problem is that of finding the correct product search form for that on-line store. Some on-line stores have several search forms, only one of which can be used to locate product information. A second sub-problem is to determine how to fill in the correct product search form, that is what product attributes, *e.g.*, product name and manufacturer, to enter into which fields in the search form. Finally, a third sub-problem is to learn how to extract the product information from the information pages returned from the search.

Solutions to these sub-problems are interdependent. A shopbot in a learning phase (hereinafter called a "learner") cannot be certain that a particular search form is appropriate until it knows how to fill it in and how to understand returned results. Therefore, the learning-phase shopbot searches combinations of solutions to these three sub-problems in order to pick the best combination.

Fig. 7 illustrates the general process for a learning-phase shopbot. Starting with URL 551 for the home page of a particular on-line store, at step 552 the learner searches at that store for candidate HTML product search forms. It thereby determines limited set 553 of candidate search forms, $F_i$ for further testing. At step 554, the learner tests each form $F_i$ to compute an estimate $E_i$ for how successful the comparison-shopping phase would be if form $F_i$ were chosen by the learner. To compute the estimate, the learner determines attribute mappings directing how to fill in the fields of the form, and then makes several "test queries," using the form to search both for several popular products of the shopping domain and also for products certain not to be in the shopping domain. The results of these test queries provide, first, training examples from which the learner determines the format of product descriptions in the pages resulting from form $F_i$, including the header, trailer, and item format strings. Second, test queries provide examples of search failure pages from which the learner determines the failure string. Third, the results are also used to compute $E_i$. An estimate of the learner's success in extracting information for these popular products provides an estimate of how well the system would do in general for ordinary products. Thus, the learner determines complete vendor descriptions 555 based on each of the candidate search forms found. Finally at step 556, the learner picks the form with the best estimate, $E_i$. The learner's final output 557 consists of a vendor description: the chosen form's URL, the failure string, the attribute mappings, the header and trailer strings, and the item format.

In the following, key steps of this process are described in greater detail. These steps depend on various heuristics. This invention is equally adaptable to alternative heuristics that achieve similar functions in similar manners.

**Finding Candidate Forms**

The first step is to find candidate search forms at the store's web site. Forms are parts of HTML formatted web pages starting with a "<form>" tag and ending with a "</form>" tag. An exhaustive search, starting at the store's home page and recursively following all HTML links, is much less preferred. Such a search could, for some stores, eventually reach most of the millions of pages on the WWW, and, second, could result in a large number of pages being fetched from the given store, which would place a heavy burden on that store's server.

To avoid these problems, it is preferred to place a limit, currently preferably between 25 and 100 and most preferably 50, on the number of pages a learner fetches while trying to find candidate forms. Given such a limit, however, the search procedure is preferably more selective. For example, if a site has 500 pages, only one of which has the right form on it, but a learner looks at only 50 of them, then a random search would have only a 10% chance of success. Therefore the learner incorporates heuristic techniques designed to increase its chances of finding the right page despite a limited search.

According to this technique, the learner prioritizes fetching of pages according to a priority scoring function. Lower scores are better, *i.e.*, considered more likely to be the page containing the right search form. A priority queue of URLs of pages to be fetched is maintained. Initially, this queue contains just the URL of the store's home page with score 0. The learner repeatedly removes the highest priority, or lowest scored, URL from the queue, fetches that page, and adds to the queue the URL's from any links contained in that page. As soon as 50 pages have been fetched, the search stops, and the learner searches for forms on the 50 pages retrieved.

No page gets fetched twice By keeping track of which pages have already been fetched. Further, the learner also has a list of forbidden domains from which it never fetches pages. Forbidden domains ar URLs that many WWW sites are

linked to, e.g., netscape.com, microsoft.com, lycos.com, altavista.digital.com, and so forth, that are known not to be part of any on-line store's WWW site.

The following pseudocode illustrates the process for finding a set of candidate forms.

```
PROCEDURE FIND_CANDIDATE_FORMS (starting_url, max_pages):
   /* Initial queue is just starting url with score 0 */
   initialize priority queue: pages_to_do = <starting_url,
   priority = 0>
   initialize set page_already_done = empty set;
   initialize set forms_found = empty set;
   /* main loop: fetch a page, add its links to queue */
   WHILE  ( (pages_to_do is not empty) AND
        (size(pages_already_done) < max_pages) ) DO BEGIN
     remove the minimum score item from pages_to_do queue;
     name this item "<url, this_score>";
     IF ( (url is in pages_already_done) OR (url accesses a
        FORBIDDEN DOMAIN) ) THEN skip this url go on to the
        next iteration of the WHILE loop;
     fetch the url; name the HTML text fetched "page";
     add url to pages_already_done;
     FOR EACH form on page, add that form to forms_found
     FOR EACH outgoing HTML link on page DO BEGIN
          LET u_link denote the url of the link and t_link
             denote the text of the link; i.e., the HTML
             code for the link has the form "<a href =
             "u_link"> t_link </a>";
          IF (t_link contains as a substring the words
             "search," "find," or a variant thereof) THEN
             score = this_score + 0.1
          ELSE IF (t_link contains a substring "text mode,"
             "text only mode," "no graphics," or a variant
             thereof)
          THEN score = this_score + 0.2;
          ELSE BEGIN
```

```
                /* see explanation of "position-component" in
                text */
                score = this_score + 1.0 + position-component;
                IF (u_link is in a different domain than url)
                THEN score = score + 2.0;
            END;
            add <u_link,score> to the priority queue
                pages_to_do
        END /* of FOR loop */
    END /* of WHILE loop */
    RETURN forms found
END /* of PROCEDURE FIND_CANDIDATE_FORMS */
```

The scoring function is designed to give priority to those URLs considered more likely to contain or to lead to the desired search form. Five heuristic rules are used to compute the score. First, the score for a given link is always higher than the score from the page containing it. The score is always computed by adding a positive number to a previous score. This is because the farther away from the store's home page, *i.e.*, the more links followed from the home page, the less likely is the search form. On-line stores usually put the search form within 2 or 3 clicks from their home page. Second, a link that contains the word "search" or "find" is likely to lead to the search form page, so it gets a relatively good score. Third, a link that points to a "text-only mode" section of the store's web site gets a good score, since a shopbot is more likely to understand text pages. Fourth, if there are multiple links on a page, the links near the beginning and end of the page are more likely to lead to a search form than links in the middle of the page. This is because many pages contain long lists of links to different sections of the store; a link to a general search form page is unlikely to occur in the middle of such a list. To "penalize" links in the middle of a page and "reward" links at the beginning or end, a "position-component" is added to the score. The position-component is

0 for the first and last link on the page; 1 for the second and next-to-last link on the page; 2 for the third and third-from-last link on the page; and so on. Finally, fifth, if a link jumps to a new domain, *i.e.*, a page from "foo.bar.com" has a link to "aaa.bbb.com", then the link is penalized, since it is likely to lead to some completely different site that isn't part of the store. However, such links are not discarded because some stores' WWW sites do in fact span multiple domains.

Filling Out A Candidate Form

Fig. 8 illustrates in greater detail the process of step 554 of Fig. 7. In general, starting with a set of candidate forms, forms 553 of Fig. 7, at step 602 a learner selects from the candidate forms those likely to be product search forms. At step 603, a learner then uses rules from the domain description to build an attribute mapping, which guides the learner is filling in form fields appropriately. Using these as input, at step 605 a learner queries the on-line store with dummy products in order to determine a failure format string, and at step 607, it queries the store with popular products in order to determine the format of product information in a successful product page. If any of these step fail, a learner abandons processing the current candidate form and starts with the next available form, if any. Once all candidate forms have been processed or abandoned, that form having the highest evaluation, along with the determined descriptive strings, is used for the description of this on-line store.

This process is described in more detail in the following paragraphs. First, starting with URL 601 giving the location of the next candidate form, a learner then determines how to fill in that candidate form, that is what product attributes, *e.g.*, product name, manufacturer, and so forth, to enter into each of the fields in the search form. To determine this, a learner first examines the HTML text of the form to extract the various type-in input fields, which

are specified by HTML <input> tags, and locates the corresponding prompts, *i.e.*, the text immediately preceding the input field which normally prompts the user concerning what data should be entered in that field.

Next, a learner checks for the presence of several conditions which indicate that the form is almost certainly not a product search form. In this case, the form is discarded at step 609 without an attempt to fill it out. Four such preferable conditions are described here. A first such condition is that submitting the form would require accessing a forbidden domain. A second such condition is that one of the type-in input fields is of type PASSWORD or TEXTAREA, as specified in attributes of the HTML tags. Such kinds of input fields are rarely found in search forms. A third such condition is that the form contains no input fields of type TEXT, *i.e.*, no ordinary type-in input fields, at all. A fourth such condition is that one of the field's prompts contains such words as "mastercard," "email," "e-mail," "phone," "telephone," and so forth. In this case, the form is probably a user registration form or an order form, and not a search form. Other such conditions can used as appropriate in various product domains or as on-line stores evolve and change.

If none of these conditions hold, then a learner proceeds to step 603 in order to determine which product attributes, *e.g.*, name, manufacturer, and so forth, are to be entered into each type-in input field. This determination is made differently for different product domains, *i.e.*, for computer software stores there is one set of attributes to choose from, such as product name, version, hardware platform, operating system, and so forth, while for music CD's there are different attributes, such as artist, album title, and so forth. In general, the determination is made using a product-domain dependent set of rules that test the prompt of a field and decide what product attributes to enter into that field. An exemplary rule might test the field's prompt for the word "name," and if found, fill in that field

with the product name. A further exemplary rule for a product domain without part numbers, might test the field's prompt for the words "part number," and if found, leave that field blank, since the learner knows nothing about part numbers. For each product domain, e.g., software, CD's, etc., the domain description must provide a leaner with a list of such rules. Then, for each input field in a candidate form, a learner system sequences through this list of rules and applies the first rule whose test matches the field's prompt. If a rule applies, an attribute mapping pair is added to the already found mapping pairs. The pair comprises the string in the field prompt that was matched by the rule paired with the name of the product attribute that the rule indicates should be entered in this fill-in-field. If none applies, the field is not filled in with anything.

Output 604 from step 603 includes the URL of the candidate form together with the constructed attribute mapping.

Learning Result Formats

Having determined the attribute mappings, which guide how to fill in a candidate form, at steps 605 and 607 a shopbot in a learning phase determines how to extract information, in other words to parse, result pages returned from the on-line store after this filled-in form has been submitted. Preferably, the learner relies on several typical regularities of the product information pages. First, for each form, the result pages typically are of two types: a "failure" type, when nothing in the store's database matched the query parameters; and a "success" type, when one or more items matched the query parameters. Second, success pages typically consist of a header, a body, and a tailer, where the header and tailer are consistent across pages from different product searches, and where the body contains all the desired product information, along with possibly irrelevant information as well. Third, the product descriptions typically have the same unique format, not

possessed by anything else in the body of the page. Using these regularities, in order to parse a result page a learner solves three additional sub-problems: first, learning the generalized failure template; second, learning to remove irrelevant header and tailer information; and third, learning product description formats.

Accordingly, at step 605 a shopbot first determines a general failure template for a form by querying the form with several "dummy" products almost certainly not in the database, such as the product named "qrsabcdummynosuchprod" from company "MadeUpManufacturerName." More specifically, the result page for one dummy product, *e.g.*, "qrsabcdummynosuchprod," is fetched and each occurrence of that dummy product name in the result page is replaced with a special placeholder, *i.e.*, replace each occurrence of the string "qrsabcdummynosuchprod" in the result page with the string "***DUMMY-NAME***", and replace each occurrence of "MadeUpManufacturerName" with "***DUMMY-MANUFACTURER***". This procedure is repeated for several more such dummy products with different names. If the result pages, after replacement of strings, are the same in every case, then the learner records this page as the failure format string for this form. If the different result pages for different dummy queries are not the same, then the learner abandons further processing with this candidate form at step 609 and goes on to the next candidate form found in set 553 of Fig. 7.

Output 606 from this step includes the URL of the form, attribute mappings for the form, and the failure format string by which search failures can be recognized. The following pseudocode illustrates the procedure for querying with dummy products.

```
initialize result_pages = empty-set;
FOR EACH ( dummy product in this domain denoted by
<attr1=value1, attr2=value2, ..., attrN=valueN> ) DO BEGIN
    page = (fill out this form using this dummy product
        and fetch the result page);
```

```
        FOR i=1 TO N DO BEGIN
            replace all occurrences of "value_i" in page
            with "***DUMMY-attr_i***";
        END;
        add page to the set result_pages;
    END;
    IF (all elements in result_pages are the same)
        THEN FAILURE_FORMAT = this one page
        ELSE skip this form and go to the next form from
        step 1;
```

Next, at step 607 in order to learn how to recognize pages returned from a successful product query, a shopbot learner queries the form with several popular products from the domain as provided in the domain description, *e.g.*, the current best-selling products in this domain. It compares each result page for these products against the failure format learned above; any page that matches the failure format is assumed to represent a failed search for this form and is discarded at step 609. If the majority of the test queries with the popular products are failures rather than successes, the learner determines that this is not the appropriate search form to use for the vendor, and it goes on to the next candidate form found in set 553. Otherwise, the learner records generalized templates for the header and tailer of success pages, by replacing words which are product attributes with fixed standard strings, and then by finding the longest matching prefixes and suffixes substrings of the success pages obtained from the test queries.

The output of this process now includes the header and trailer strings by which the uninformative portions of a successful product query page can be recognized and discarded. The following pseudocode illustrates this process.

```
    initialize result_pages = empty-set;
    initialize result_copies = empty-set;
```

```
FOR EACH ( test product in this domain denoted by
<attr1=value1, attr2=value2, ..., attrN=valueN> ) DO BEGIN
     page = (fill out this form using this dummy product
     and fetch the result page);
     copy_of_page = page;
     FOR i=1 TO N DO BEGIN
          replace all occurrences of "valuei" in
          copy_of_page with "***DUMMY-attri***"
     END;
     IF ( copy_of_page is different from FAILURE_FORMAT)
          THEN add page to the set result_pages;
END;
IF ( size(result_pages) < the number of test products)
     THEN give up on this form and go on to the next
     form from step 1;
set HEADER = longest common prefix substring of all the
     result_copies;
set TAILER = longest common suffix substring of all the
     result_copies;
```

Continuing in step 607, a learner now uses the bodies of the pages from successful searches as training examples from which to determined the format of product descriptions in the result pages for this form. Each such page contains one or more product descriptions, each containing information about a particular product, or version of a product, that matched the query parameters. The format of these product descriptions varies widely across vendors. However, at each particular vendor, all the product descriptions usually have the same abstract format. The learning phase processing searches through the possible abstract formats and picks the best one, *i.e.*, the one it determines to be most likely to correspond to product descriptions at this site.

The abstract formats are described by strings of HTML tags together with keyword "text". The abstract form of a fragment of HTML is obtained by removing the arguments from HTML tags and replacing all occurrences of intervening text

with the keyword "text." For example, the HTML source string "<li>Click<a href="http://store.com/Encarta">here</a>for Encarta." is abstracted into the abstract form string "<li>text<a>text</a>text."

There are many abstract formats to consider in the search, *i.e.*, many possible sequences of HTML tag names and "text". Even considering only the finitely many which actually occur in one of the bodies of the success pages from the test products, there is still a large number of possible formats to consider. So this number is further reduced by assuming that every product description starts on a fresh line, as specified by certain HTML tags such as <p>, <br>, <li>, etc. Therefore, the process first breaks the body of each result page into logical lines which are strings separated by vertical-space-delimited HTML tags, and only then considers abstract formats that correspond to at least one of the logical lines in one of the result pages.

The bodies of success pages typically contain logical lines with a wide variety of abstract formats, only one of which corresponds to product descriptions. The learner uses a heuristic ranking process to choose which format is most likely to be the one the store uses for product descriptions. A preferred ranking function is the sum of the number of logical lines of that format in which some text, not just white space, was found, plus the number of logical lines of that format in which a price was found, plus the number of logical lines in which one or more of the required attributes were found. This heuristic exploits the fact that since the test queries are for popular products, on-line stores tend to stock multiple versions of each product, leading to a plurality of product descriptions on a successful page. This ranking function reflects both the number of popular products that were found and the amount of information present about each one. The exact details of the heuristic ranking function do not appear to be crucial, since there is typically a large disparity between the rankings of the "right" format and alternative "wrong" formats. This

invention is adaptable to other heuristic ranking functions that achieve similar discriminations.

Final output 608 of step 607 includes all the components of an on-line store description built from the current candidate form as well as the value of the ranking function. If the form is to be abandoned as described for previous steps, this ranking is set to a large negative value so that it will be ignored in the further processing. This last process of step 607 is illustrated in the following pseudocode.

```
lines = empty set;
formats = empty set;
FOR EACH page in result_pages DO BEGIN
     Let body be the substring of page remaining after
          removing the prefix and suffix substrings
          matching header and tailer;
     Divide the body string into a set of logical lines,
          or substrings, at occurrences of vertical-
          space-delimiting HTML tags. such as <p>, <br>,
          etc;
     FOR EACH logical line DO BEGIN
          Add this line to the set lines;
          Add abstract version of this line to the set
               formats;
     END;
END;
RETURN    (the format in formats which maximizes
          evaluate ranking function)

FUNCTION evaluate (format):
     INTEGER n = 0;
     BEGIN
          FOR EACH line in lines FOR WHICH (abstract
               version of line)=format DO BEGIN
               IF (line contains any text other than
                    white space) THEN n=n + 1;
```

```
                IF (line contains a price, i.e., a dollar
                    sign followed by digits) THEN n=n +
                    1;
                IF (line contains any of the test product
                    value_i's as a substring) THEN n=n +
                    1;
            END;
        RETURN (n);
    END
```

Generating the Vendor Description

Finally, a learner must chose the candidate form to use for this on-line store. Returning to Fig. 7, the learner has processed each candidate form found in set 553 according to the previously described steps illustrated in Fig. 8. For each form, it determines how to fill in the query attributes, and how to parse the result pages to find a best abstract format, that is the format that maximizes the function evaluate(format). At step 556, it chooses one of those forms with the greatest value of the ranking function for shopping at this on-line store. As mentioned above, this choice is based on making an estimate $E_i$ for each form $F_i$ of how successful the shopping would be if form $F_i$ were chosen by the learner. The $E_i$ used is the value of the evaluate function for the winning abstract product description format. This function reflects both the number of the popular products that were found and the amount of information present about each one. Thus the form selected is the one whose best format has the greatest value of evaluate among all other candidate forms, and thus the greatest use in accessing this on-line store.

Once the learning phase has chosen a form, it records a vendor description (See Table 2) for future use by the shopping phase. If the learner is unable to find any form that yields a successful search on a majority of the popular products, then shopbot abandons this vendor.

The learning phase runs once per merchant prior to any shopping at this merchant. The learner's running time is linear in the number of vendors, the number of forms at a vendor's site, the number of "test queries," and the number of lines on the result page. The learner typically takes 5-15 minutes per vendor.

## 5.4. SHOPPING AT ON-LINE STORES

TABLE 2 - A VENDOR DESCRIPTION

| |
|---|
| o The URL of a page containing a product search form, and optionally an encoding of the preferred search form on this page<br>o Pairs of strings, each pair being an attribute name and a field name, for mapping of product query attributes to fields of that form.<br>o Strings used by parsing functions for extracting product data from pages returned from the on-line store by matching portions of the returned pages, including<br>- A string that matches a unique portion of a search failure page (e.g., "Product not found").<br>- Header and trailer strings that match mere header and trailer formatting information used in search success pages.<br>- An abstract format string that matches the components of product descriptions in search success pages and is used to extract information from those descriptions. |

The output of the first learning phase is an on-line store description, which together with the domain description is used in the second comparison-shopping phase. Table 2 lists the preferred components of an on-line store description. In a preferred embodiment, the output can be several strings used by the functions subsequently described for comparison shopping. These strings include location string, attribute mappings, failure string, header and trailer strings, and an item format string. The location string contains the URL of the WWW page containing the product search form. If there are multiple forms on the search page, the location string additionally encodes which

form to use for product searching. The attribute mappings are pairs of strings of the form ("attribute-name", "field-name"). The "attribute names" are names of the attributes defined in the domain description, *e.g.*, "title", "manufacturer", "artist" for a CD store domain. The "field names" are labels for the fields on the search form to be filled in with values for the corresponding "attribute name." HTML requires a label for each fill-in input field. The failure format is a single string matching a distinctive portion of the search failure page returned from the on-line store if a product search fails. The header and tailer strings match the header and trailer formatting portions of a successful product search page returned from the on-line store. Finally, the item format is a single string which matches an abstract version of each "logical line" of product information returned on a successful product search page. An abstract version of a logical line has all text, that is everything other than HTML keywords, replaced by the string "text." This invention is adaptable to other formats for vendor descriptions. For example, it is adaptable to a language based on regular expressions for parsing HTML formatted documents.

During the second comparison-shopping phase, the vendor and domain descriptions are used to assist a user in comparison-shopping at an on-line electronic store on the Internet. In a preferred embodiment in which the vendor description contains the above described strings, this phase proceeds generally as illustrated in Fig. 9. By means of user interface 650, a user inputs a product information request to a shopbot, which can be, for example, purchase request 655 to find the cheapest price for a product. Shopping phase modules 651 proceed generally according to the process illustrated in Fig. 9. Domain description 653 for the product query is retrieved from the store/domain description database. The user request is represented as the values of the product attributes appropriate to this domain as identifi d in the domain description. Next, shopbot 651

retrieves the vendor descriptions for relevant on-line stores having products in this domain. Guided by vendor descriptions 654, for each vendor, shopbot 651 access that on-line stores's product search form, fills in the form with the query attributes, and submits filled-in form 656 to the on-line store. The store returns product information result pages 657 from which shopbot 651 parses responsive product information. These results are presented to the user by user interface 650 as, for example, best buys 658 for the product of interest. In more detail shopbot 651 process according to the following pseudocode. The input to this procedure are values of the attributes supplied by the user which defines the user's current comparison-shopping request.

```
PROCEDURE shop_for_item (<attr1=value1, attr2=value2, ...,
attrn=valuen>):
BEGIN
     Initialize the set results = empty set;
     form = fetched HTML page specified by the location
          string;
     Let "mapping" denote the learned attribute-to-field
          mapping;
     FOR EACH input field with name f in form DO BEGIN
          Fill in field f with the provided value of the
               attribute whose name corresponds to f in
               the attribute mapping pairs
     END;
     Submit the filled-in form and get the result_page;
     temp = result_page;
     FOR i=1 TO N DO BEGIN
          replace all occurrences of "valuei" in temp
               with "***Dummy-attri***"
     END;
     / *search failed, return empty set for results */
     IF temp = the learned failure format THEN EXIT
     Delete the first (length of learned header)
          characters of page;
```

```
        Delete the initial and terminal portions of
            result_page matching the header and tailer
            strings, respectively;
        Divide the remaining result_page into a set of
            logical lines (substrings) by separated by
            occurrences of <p>, <br>, etc;
        FOR EACH of these logical line DO BEGIN
            Make abstract version of logical line by
                replacing everything other than HTML
                keywords with the string "text";
            IF the abstract = the learned item format
                string THEN add this-line to results
        RETURN results
    END; /* shop */
```

Immediately after a page is fetched and before any further processing, either in the learning or shopping phases, it is checked for any graphics that can be replaced with text. Such replaceable graphics is specified in HTML by using an <IMG> tag with an "ALT" argument, which gives the text replacement. In other words the HTML keyword <IMG ... ALT="some text here"...> is replaced by just "some text here."

The shopbot stores vendor descriptions, which are a repository of knowledge about on-line stores accessible on the Internet, in a description database. Since a comparison-shopper uses the shopper to comparison shop at an on-line store only after the learning phase has been completed at that store, a shopbot is able to immediately access on-line stores and quickly search for a desired product. The shopbot shopper is both methodical and effective at actually finding products at a given vendor.

## 6. EXAMPLES

The invention is further described in the following example which is in no way intended to limit the scope of the invention.

## 6.1. SHOPBOT TRIALS

Evaluating ShopBot Utility

In a first experiment, shopbot usefulness was measured by comparison with manual comparison shopping. Seven subjects were chosen who were novices at electronic shopping but who did have experience using Netscape Navigator. The subjects were divided into three groups:

1. Three subjects who used ShopBot;
2. Two subjects who used Netscape Navigator's search tools and who were given the URLs of twelve software stores used by shopbot; and
3. Two subjects who were limited to Netscape Navigator's search tools.

Two independent parties suggested four popular software products for which to comparison shop. The products were Netscape Navigator, Hummingbird eXceed for Windows, Microsoft Word, and Intuit Quicken for the Macintosh. All subjects tried to find quickly the best price for all four of these products, and to report how long the search required. Table 3 presents the mean time and prices found for each group of subjects. Each trial of subjects in Group 1 using a shopbot were run separately and independently of each other, in order to avoid any positive effects from caching or negative effects from overloading the ShopBot server.

TABLE 3 - SHOPBOT TRIALS

| Group | Time (min:sec) | Navig ator | eXceed | Word | Quicken |
|---|---|---|---|---|---|
| 1 | 13:20 | $30.71 | $373.06 | $282.71 | $42.95 |
| 2 | 112:30 | 38.21 | (not found) | 282.71 | 41.50 |
| 3 | 58:30 | 40.95 | 610.00 | 294.97 | 42.95 |

The shopbot group completed its search task much faster than the other subjects, and generally found prices at least as low as found by the other groups. The group 3 subjects limited to Netscape Navigator's search methods never found a lower price than ShopBot users. Providing the list of store

URLs actually slowed the subjects down. For example, one group 2 subject failed to find a price for eXceed, and the other found a low price on an inappropriate version. These trials demonstrate a shopbot's utility for comparison shopping.

**Acquisition of New Software Vendors**

To assess the generality of the shopbot learning phase, an independent party not familiar with shopbot processes found ten on-line stores that sell popular software products and that have a search index at their WWW site. With the heuristics of the preferred embodiment, a shopbot was able to learn how to comparison shop at all ten vendors. Shopbot currently shops at twelve software vendors, the aforementioned ten plus two more that were used to guide the original design. This demonstrates the generality of shopbot's architecture and learning processes and heuristics within the software domain.

In more detail, Table 4 shows the line descriptions and heuristic rankings found during learning product description formats for two software vendors. In both cases, a shopbot picked the correct line description corresponding to product descriptions. Other vendors with different product formats have also been consistently learned.

TABLE 4 - FORMATS AND RANK

| Internet Shopping Network (http://www.internet.net) | | NECX Direct (http://necxdirect.necx.com) | |
|---|---|---|---|
| Line Description | Rank | Line Description | Rank |
| <br><a>*text*</a>*text* (correct format) | 324 | <li><a>*text*</a></li>*text* <br> (correct format) | 402 |
| <a>*text*</a>*text* | 4 | <h4>*text*<a>*text*</a> | 21 |
| <h2>*text*<a>*text*</a> *text* | 3 | <li><a>*text*</a></li>*text* | 12 |
| </h2>*text* | 0 | <a>*text*</a> | 4 |
| <br>*text* | 0 | <br> | 0 |
| | | </h4> | 0 |
| | | *text* | 0 |

Generality Across Product Domains

To date, a shopbot has been tested in the domains of CD's as well as software products. A domain definition that enables a shopbot to shop at pop/rock on-line CD stores has been defined. The CD domain was chosen to demonstrate the versatility and scope of shopbot's learning processes. With one day of work on describing the CD domain, a shopbot was able to shop successfully at four CD stores.

## 7. SPECIFIC EMBODIMENTS, CITATION OF REFERENCES

The present invention is not to be limited in scope by the specific embodiments described herein. Indeed, various modifications of the invention in addition to those described herein will become apparent to those skilled in the art from the foregoing description and accompanying figures. Such modifications are intended to fall within the scope of the appended claims.

Various publications are cited herein, the disclosures of which are incorporated by reference in their entireties.

WHAT IS CLAIMED IS:

1. An apparatus to assist a user in querying for information about a product at one or more on-line electronic stores, said apparatus comprising:

a. a first memory means for storing product domain descriptions;

b. a second memory means for storing descriptions of said on-line stores;

c. a first processor means for: (i) receiving a user query for said product; (ii) fetching a product domain description for the domain of said product and for fetching on-line store descriptions of one of more on-line stores for said product domain; (iii) accessing a product query form at said on-line stores according to said on-line store descriptions; (iv) filling-in said product query forms according to said domain description and said on-line store descriptions and for submitting said filled-in query forms to said on-line stores; (v) receiving response pages from said on-line stores; (vi) extracting product information from said response pages according to said on-line store descriptions; and (vii) presenting said extracted information to said user.

2. The apparatus of claim 1 wherein said product domain is the domain of computer software and hardware products or the domain of music CDs.

3. The apparatus of claim 1 wherein said first processor means is two or more processor means interconnected by a network.

4. The apparatus of claim 3 wherein one of said two or more processor means interfaces to said user by receiving said query and by presenting said extracted information which is communicated from the remaining processor means, and wherein

the remaining processor means queries said on-line stores and extracts returned product information.

5. The apparatus of claim 1 wherein said product domain description comprises a list of attributes of products in said product domain, said attributes being possible product query parameters.

6. The apparatus of claim 1 wherein said on-line store description for an on-line store comprises:

a. a network address for said product query form of said on-line store;

b. a mapping of product attributes to fill-in fields of said product query form;

c. a first function for recognizing product search failure pages from said on-line store;

d. one or more second functions for recognizing header information and trailer information of successful product search pages from said on-line store; and

e. a third function for recognizing and extracting product information from said successful product search pages.

7. An apparatus according to claim 6 wherein said network address, said mapping, said first function, said one or more second functions, and said third function are represented in said on-line store description by character strings stored in said second memory means.

8. An apparatus according to claim 7 wherein said character string for said third function comprises a sequence of HTML tags and the symbol "text" stored in said second memory means.

9. The apparatus according to claim 1 further comprising a second processor means for determining said description of an on-line store by: (i) fetching a plurality of pages from said

on-line store; (ii) selecting a plurality of candidate query forms from said plurality of pages; (iii) for each candidate query form determining a mapping of product attributes to fill-in fields of each candidate query form, querying said on-line store with said query form filled-in with dummy products and with popular products to further determine said first function, said one or more second functions, said third function, and estimating ranking of the query success of this query form; (iv) selecting the candidate query form and associated mapping and functions with the highest ranking as the on-line store description; and (v) storing said selected on-line store description in the second memory means.

10. The apparatus according to claim 9 wherein said domain description for the domain of said on-line store comprises a plurality of rules for guiding said determining a mapping of product attributes to fill-in fields.

11. A method for assisting a user in querying for information about a product at one or more on-line electronic stores, said method comprising the steps of:

a. receiving a user query for said product;

b. fetching an on-line store descriptions of said on-line stores for said product domain;

c. accessing a product query form at said on-line stores according to said on-line store descriptions;

d. filling-in said product query forms according to said on-line store descriptions;

e. submitting said filled-in query forms to said on-line stores;

f. receiving response pages from said on-line stores;

g. extracting product information from said response pages according to said on-line store descriptions; and

h. presenting said extracted information to said user.

12. The method of claim 11 wherein said steps of receiving a user query and presenting said extracted information are

performed by a user interface computer, and wherein the remaining steps of said method are performed on one or more computers that are linked among themselves and with the user interface computer by a network.

13. The method of claim 11 wherein said steps of accessing and submitting are performed in parallel for all said on-line stores.

14. The method of claim 11 wherein each of said on-line store descriptions comprises:

a. a first string representing the network address for said product query form of said on-line store;

b. a plurality of pairs of strings for mapping of product attributes to fill-in fields of said product query form, one string of each said pair being the name of a product attribute and the other string of each said pair being the name of a fill-in field of said product query form;

c. a second string for matching product search failure pages from said on-line store;

d. a third string for matching header information on successful product search pages from said on-line store;

e. a fourth string for matching trailer information of said successful product search pages; and

f. a fifth string for matching and extracting product information from said successful product search pages, said fifth string representing the abstract format of said product information.

15. The method of claim 14 further comprising prior to said step of fetching the additional steps of:

a. fetching a plurality of pages from an on-line store without a current on-line store description;

b. selecting a plurality of candidate query forms from said plurality of pages;

c. for each candidate query form determining a mapping of product attributes to fill-in fields of each candidate

query form, querying said on-line store with said query form filled-in with dummy products and with popular products to further determine said first, second, third, fourth, and fifth strings and said plurality of pairs of strings, and estimating ranking of the query success of this query form; and

d. selecting the candidate query form and associated strings and pairs of strings with the highest ranking as the on-line store description; and

16. The method of claim 15 wherein said abstract format of said product information comprises a plurality of logical lines, and wherein said fifth string matches said logical lines by being an abstract format of said logical lines.

17. The method according to claim 16 wherein said querying with said popular products returns said product search success pages, and wherein said estimating is according to the number of logical lines matched by said fifth string and having extracted text information, price information, or popular product information.

18. The method of claim 15 wherein said querying with said dummy products returns said product search failure pages, and wherein second string matches said product search failure pages.

19. The method of claim 15 further comprising prior to step (a) a step of fetching a product domain description, said product domain description comprising one or more rules for guiding said determining a mapping of product attributes.

20. The method of claim 15 wherein said selecting a plurality of candidate query forms selects one of said plurality of pages as a candidate query form according to whether said page is near the home page of said on-line

store, contains a link likely to lead to a search page, or whether said page is in the same domain as said home page.

21. A computer readable medium for causing a processor to function according to the method of claim 11.

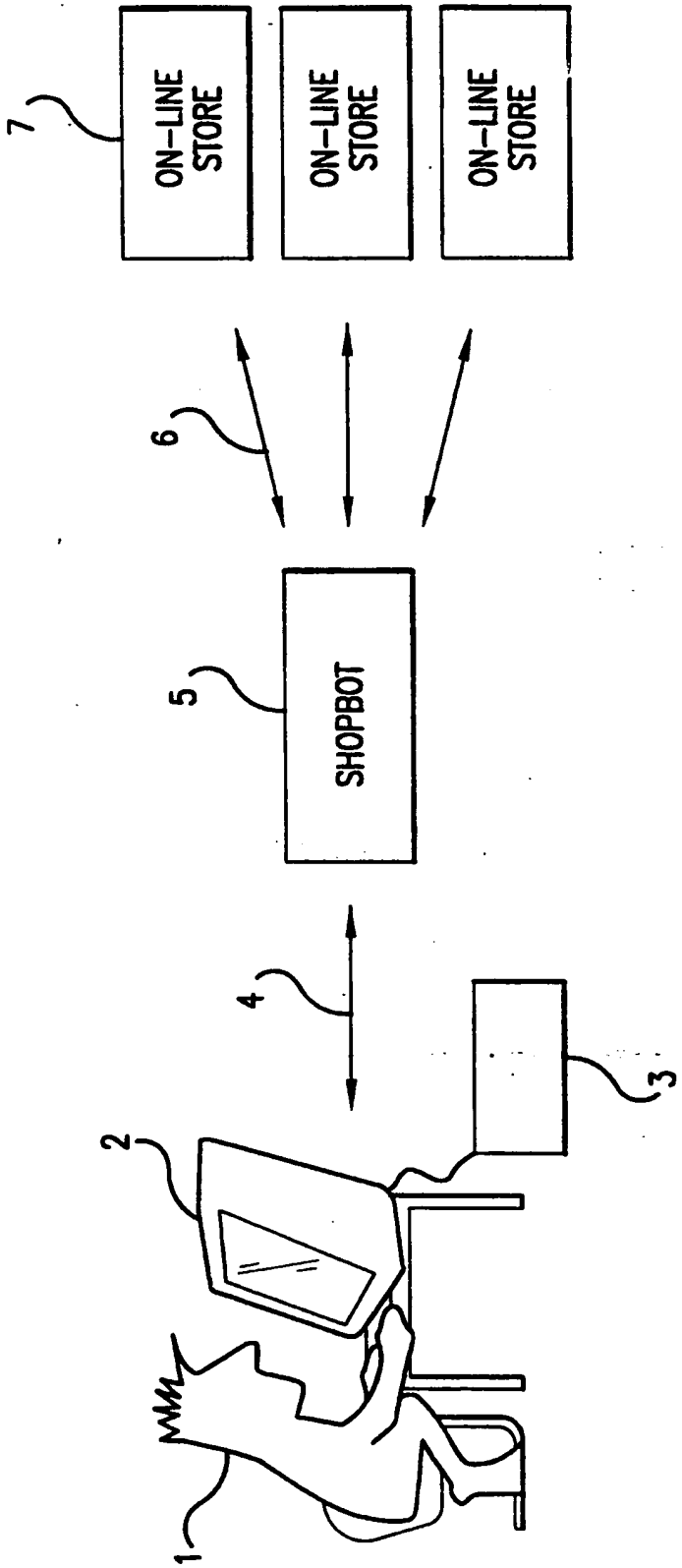



FIG.1

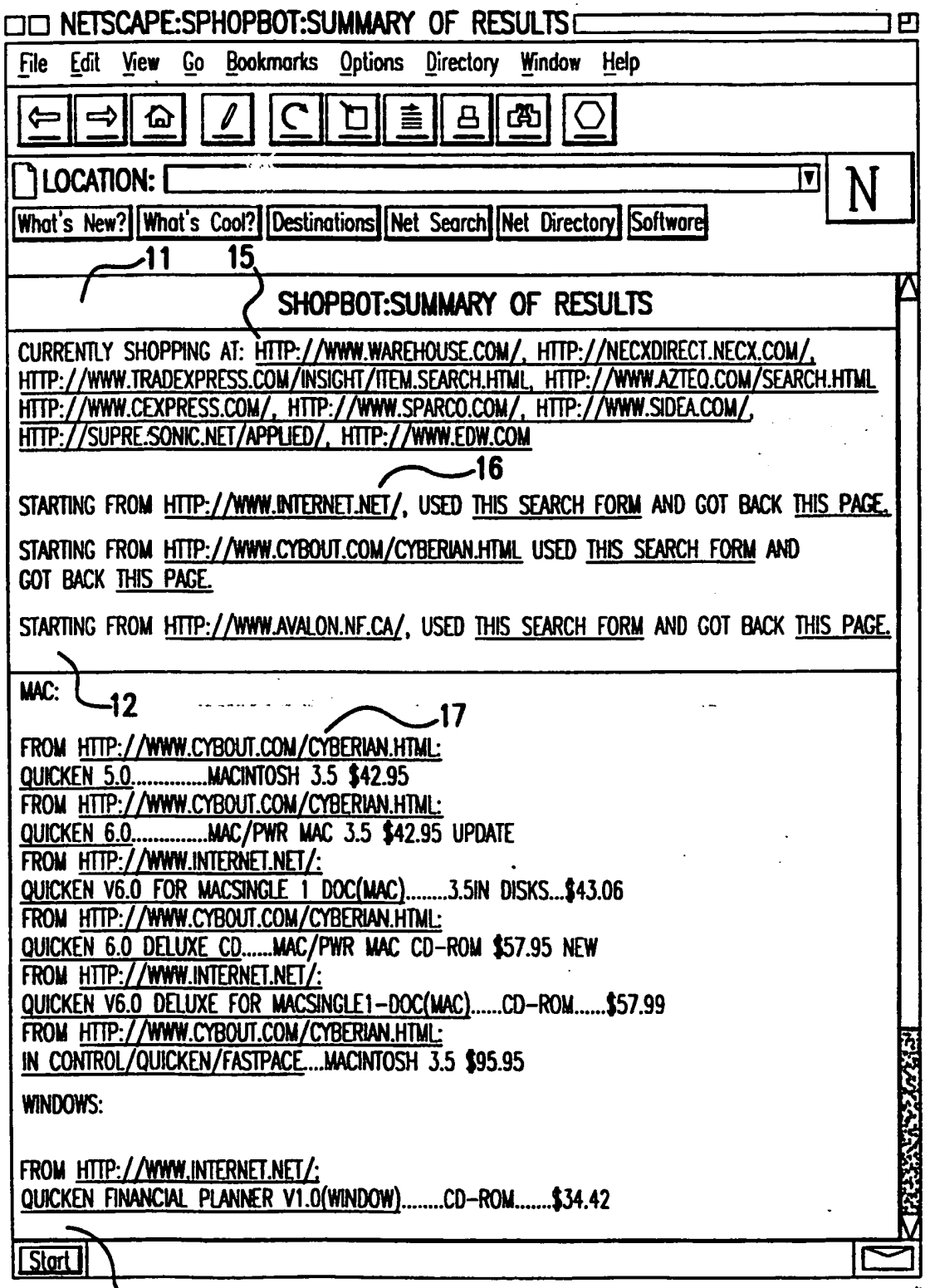

NETSCAPE:SPHOPBOT:SUMMARY OF RESULTS
File Edit View Go Bookmarks Options Directory Window Help
LOCATION:
N
What's New? What's Cool? Destinations Net Search Net Directory Software
11
15
SHOPBOT:SUMMARY OF RESULTS
CURRENTLY SHOPPING AT: HTTP://WWW.WAREHOUSE.COM/, HTTP://NECXDIRECT.NECX.COM/, HTTP://WWW.TRADEXPRESS.COM/INSIGHT/ITEM.SEARCH.HTML, HTTP://WWW.AZTEQ.COM/SEARCH.HTML HTTP://WWW.CEXPRESS.COM/, HTTP://WWW.SPARCO.COM/, HTTP://WWW.SIDEA.COM/, HTTP://SUPRE.SONIC.NET/APPLIED/, HTTP://WWW.EDW.COM
16
STARTING FROM HTTP://WWW.INTERNET.NET/, USED THIS SEARCH FORM AND GOT BACK THIS PAGE.
STARTING FROM HTTP://WWW.CYBOUT.COM/CYBERIAN.HTML USED THIS SEARCH FORM AND GOT BACK THIS PAGE.
STARTING FROM HTTP://WWW.AVALON.NF.CA/, USED THIS SEARCH FORM AND GOT BACK THIS PAGE.
MAC:
12
17
FROM HTTP://WWW.CYBOUT.COM/CYBERIAN.HTML:
QUICKEN 5.0.....MACINTOSH 3.5 $42.95
FROM HTTP://WWW.CYBOUT.COM/CYBERIAN.HTML:
QUICKEN 6.0.....MAC/PWR MAC 3.5 $42.95 UPDATE
FROM HTTP://WWW.INTERNET.NET/:
QUICKEN V6.0 FOR MACSINGLE 1 DOC(MAC).....3.5IN DISKS...$43.06
FROM HTTP://WWW.CYBOUT.COM/CYBERIAN.HTML:
QUICKEN 6.0 DELUXE CD.....MAC/PWR MAC CD-ROM $57.95 NEW
FROM HTTP://WWW.INTERNET.NET/:
QUICKEN V6.0 DELUXE FOR MACSINGLE1-DOC(MAC).....CD-ROM.....$57.99
FROM HTTP://WWW.CYBOUT.COM/CYBERIAN.HTML:
IN CONTROL/QUICKEN/FASTPACE....MACINTOSH 3.5 $95.95
WINDOWS:
FROM HTTP://WWW.INTERNET.NET/:
QUICKEN FINANCIAL PLANNER V1.0(WINDOW).....CD-ROM.....$34.42
Start
13

FIG.2

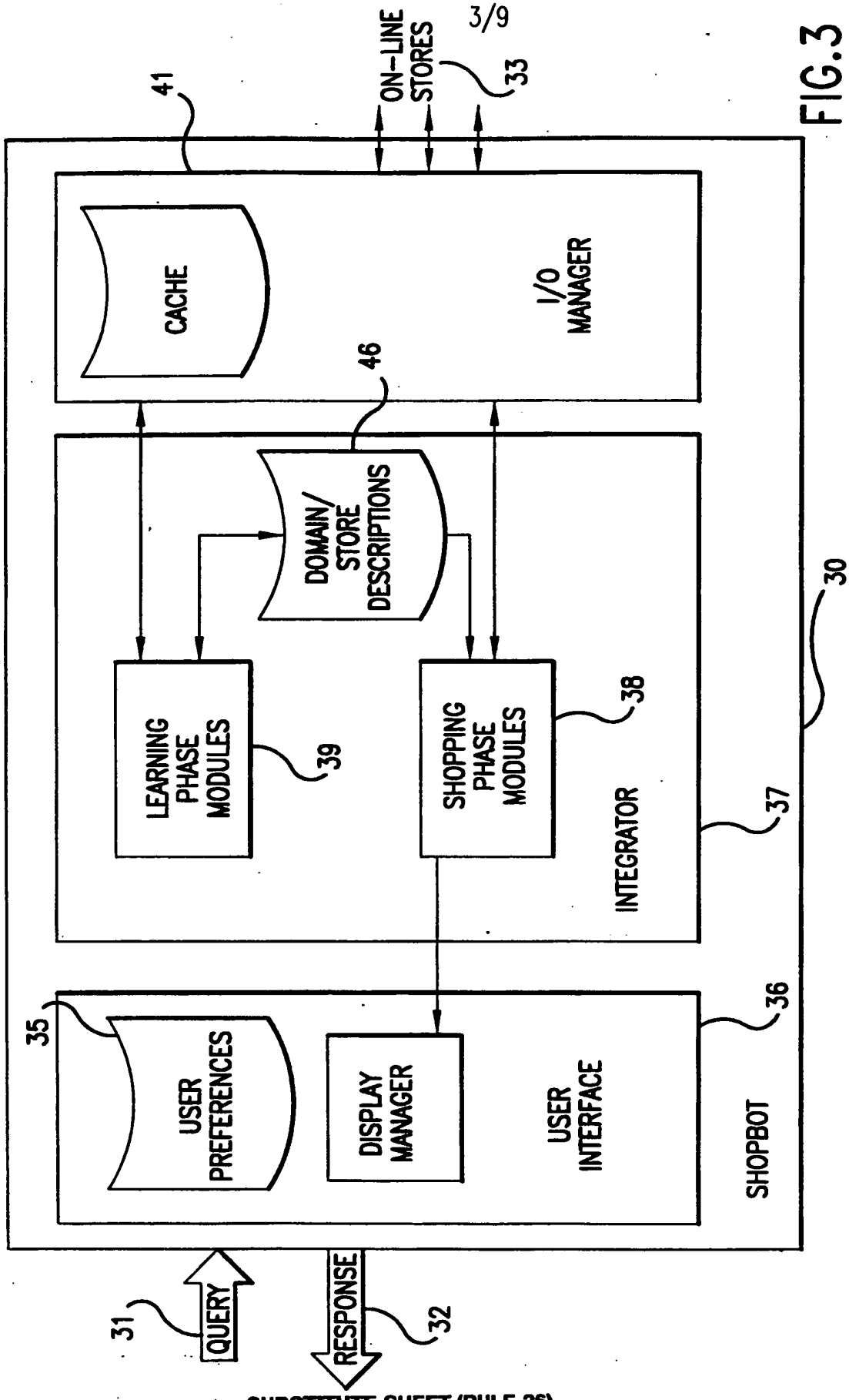

ON-LINE STORES
33
41
CACHE
I/O MANAGER
46
DOMAIN/ STORE DESCRIPTIONS
LEARNING PHASE MODULES
39
SHOPPING PHASE MODULES
38
INTEGRATOR
37
30
35
USER PREFERENCES
DISPLAY MANAGER
USER INTERFACE
36
SHOPBOT
31
QUERY
RESPONSE
32

FIG.3

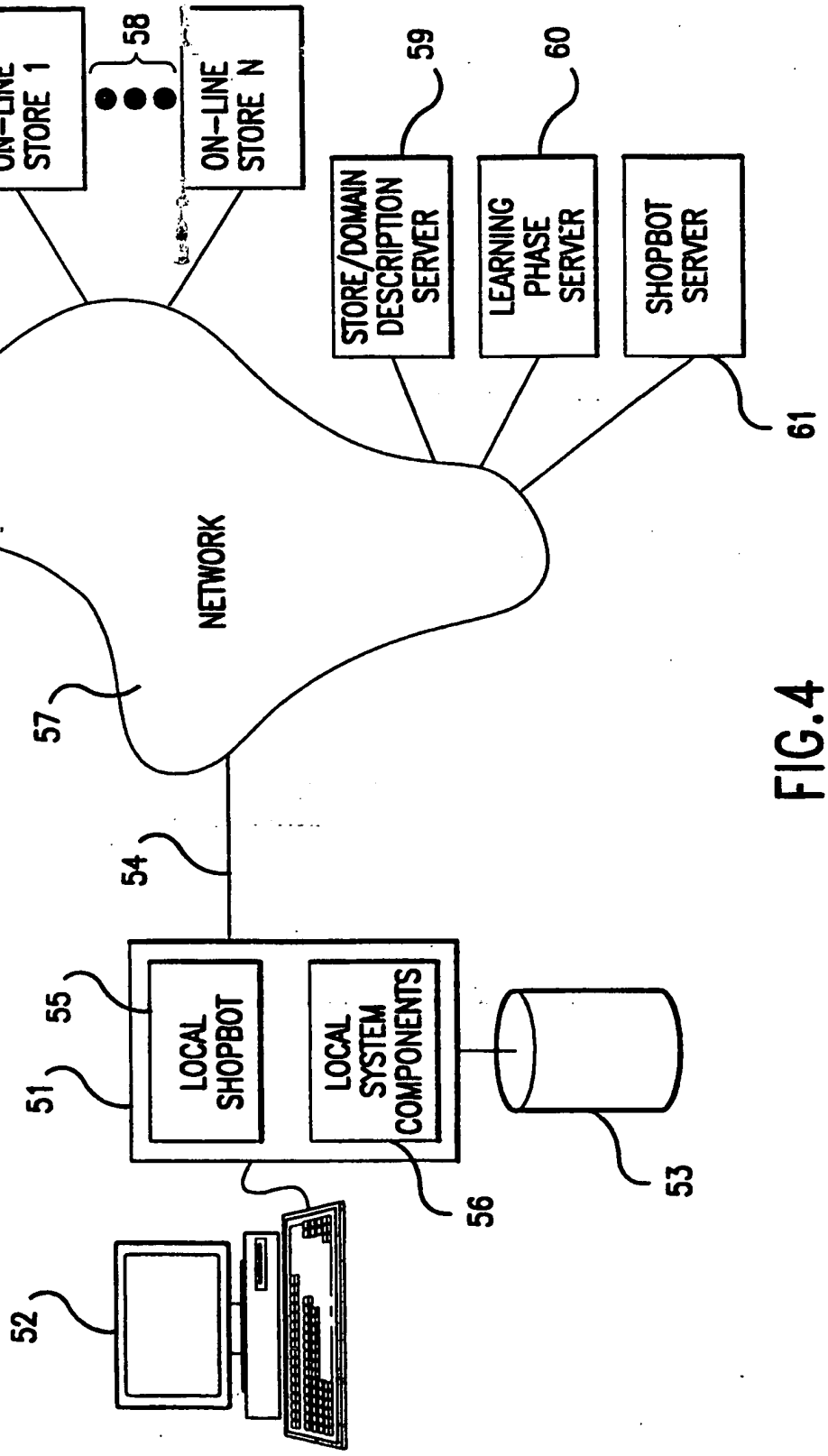

ON-LINE STORE 1
58
ON-LINE STORE N
59
STORE/DOMAIN DESCRIPTION SERVER
60
LEARNING PHASE SERVER
SHOPBOT SERVER
61
NETWORK
57
54
55
LOCAL SHOPBOT
51
LOCAL SYSTEM COMPONENTS
56
53
52

FIG.4

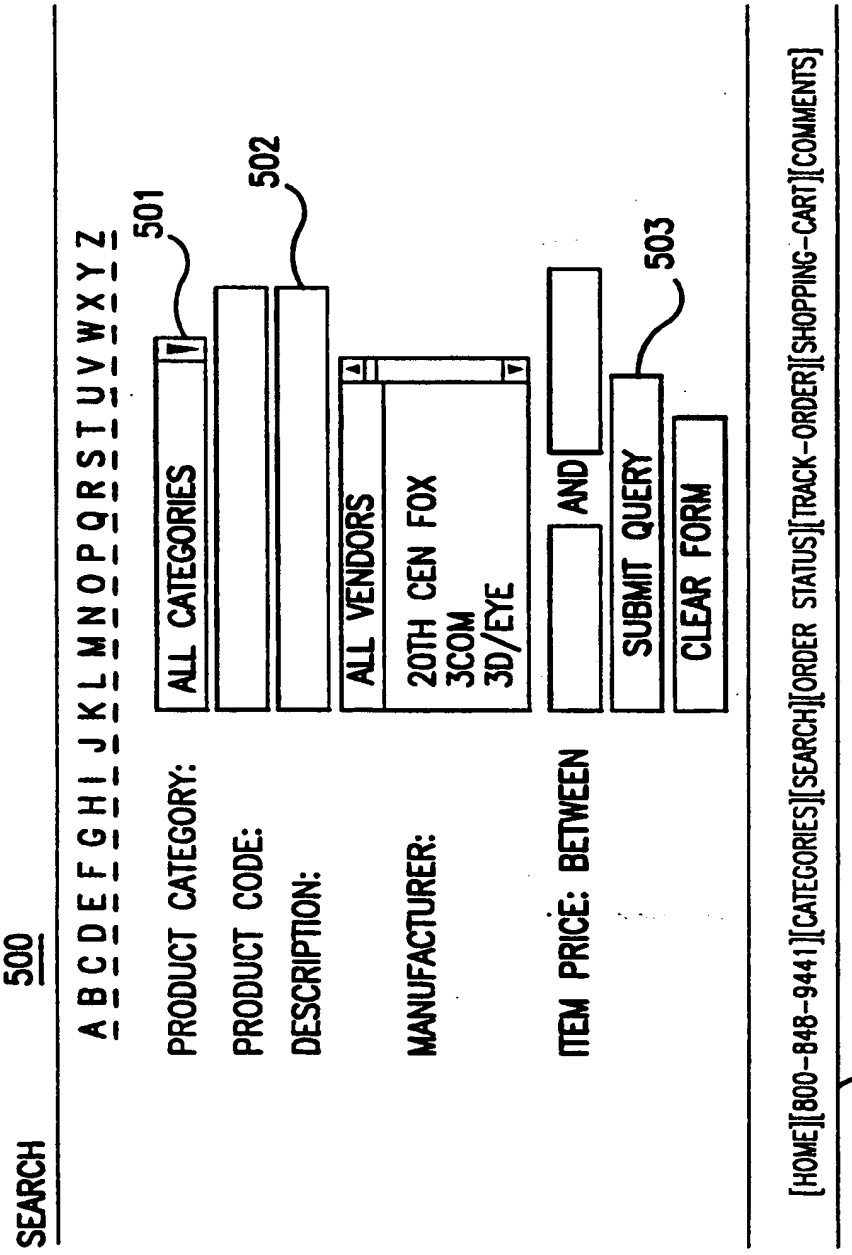

SEARCH
500
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
PRODUCT CATEGORY:
ALL CATEGORIES
501
PRODUCT CODE:
502
DESCRIPTION:
MANUFACTURER:
ALL VENDORS
20TH CEN FOX
3COM
3D/EYE
ITEM PRICE: BETWEEN
AND
SUBMIT QUERY
503
CLEAR FORM
[HOME][800-848-9441][CATEGORIES][SEARCH][ORDER STATUS][TRACK-ORDER][SHOPPING-CART][COMMENTS]
504

FIG.5

507 ITEM SEARCH RESULTS 505

IOJAZ1GBC3
3-PACK OF IOMEGA JAZ 1GB CARTRIDGES - PC
PRICE: 299.95
STOCK STATUS: IN STOCK 506

IOJAZ1GBM
IOMEGA JAZ CARTRIDGE 1GB FOR MAC
PRICE: 129.00 [TECH INFO]
STOCK STATUS: IN STOCK 506

IOJAZCASE
IOMEGA JAZ CARRYING CASE FOR THE JAZ DRIVE
PRICE: 29.95
STOCK STATUS: IN STOCK

IOJAZINT
IOMEGA JAZ INTERNAL 1GB SCSI REMOVABLE CARTRIDGE HARD DRIVE
PRICE: 399.95 [TECH INFO]
STOCK STATUS: IN STOCK

IOJAZJET2
IOMEGA JAZ JET SCSI HOST PCI ADAPTER FOR PC
PRICE: 99.95 [TECH INFO]
STOCK STATUS: IN STOCK

IOJAZTRAVL
IOMEGA JAZ TRAVELLER SCSI TO PARALLEL CONVERTER 50HD>BD25
PRICE: 44.99
STOCK STATUS: IN STOCK

[800-848-9441][HOME][SEARCH][ORDER STATUS][TRACK][SHOPPING-CART][ON-LINE CATALOG]

PLEASE EMAIL WEBMASTER@INSIGHT.COM WITH ANY PROBLEMS, SUGGESTIONS OR QUESTIONS.

508

FIG.6

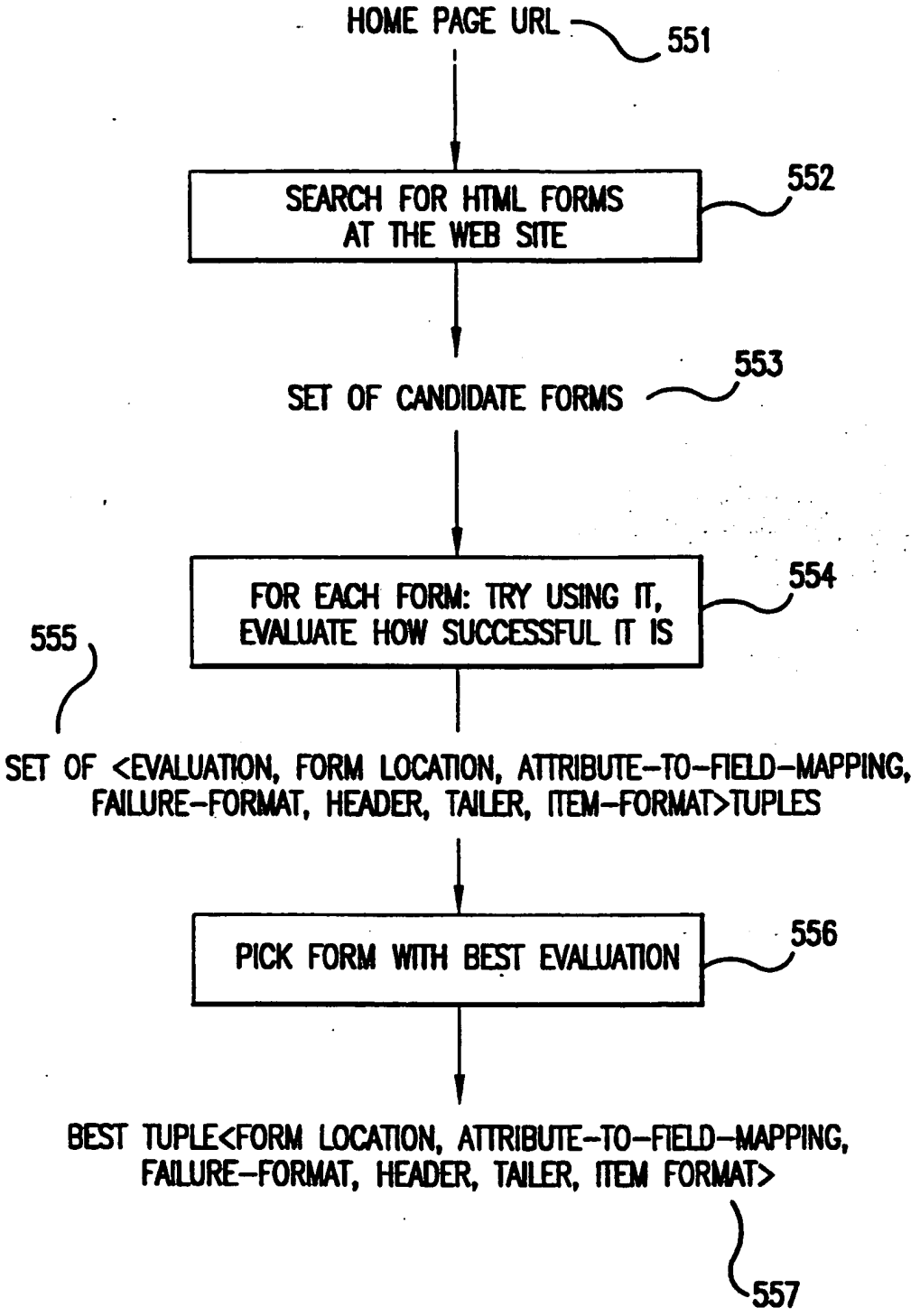

HOME PAGE URL
551
SEARCH FOR HTML FORMS AT THE WEB SITE
552
SET OF CANDIDATE FORMS
553
FOR EACH FORM: TRY USING IT, EVALUATE HOW SUCCESSFUL IT IS
554
555
SET OF <EVALUATION, FORM LOCATION, ATTRIBUTE-TO-FIELD-MAPPING, FAILURE-FORMAT, HEADER, TAILER, ITEM-FORMAT>TUPLES
PICK FORM WITH BEST EVALUATION
556
BEST TUPLE<FORM LOCATION, ATTRIBUTE-TO-FIELD-MAPPING, FAILURE-FORMAT, HEADER, TAILER, ITEM FORMAT>
557

FIG.7

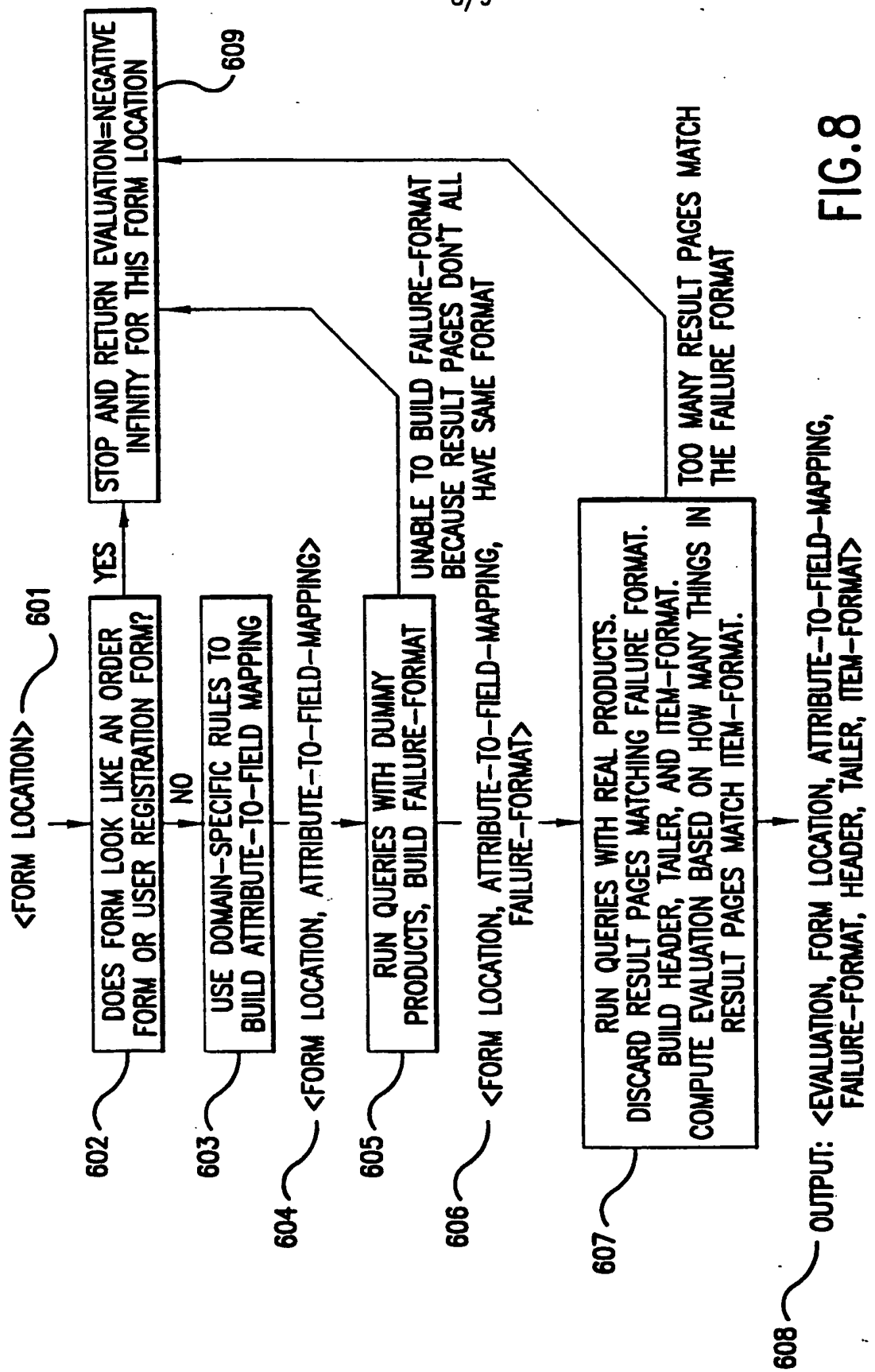

<FORM LOCATION>
601
DOES FORM LOOK LIKE AN ORDER FORM OR USER REGISTRATION FORM?
YES
NO
602
STOP AND RETURN EVALUATION=NEGATIVE INFINITY FOR THIS FORM LOCATION
609
USE DOMAIN-SPECIFIC RULES TO BUILD ATTRIBUTE-TO-FIELD MAPPING
603
<FORM LOCATION, ATTRIBUTE-TO-FIELD-MAPPING>
604
RUN QUERIES WITH DUMMY PRODUCTS, BUILD FAILURE-FORMAT
605
UNABLE TO BUILD FAILURE-FORMAT BECAUSE RESULT PAGES DON'T ALL HAVE SAME FORMAT
<FORM LOCATION, ATTRIBUTE-TO-FIELD-MAPPING, FAILURE-FORMAT>
606
RUN QUERIES WITH REAL PRODUCTS.
DISCARD RESULT PAGES MATCHING FAILURE FORMAT.
BUILD HEADER, TAILER, AND ITEM-FORMAT.
COMPUTE EVALUATION BASED ON HOW MANY THINGS IN RESULT PAGES MATCH ITEM-FORMAT.
607
TOO MANY RESULT PAGES MATCH THE FAILURE FORMAT
OUTPUT: <EVALUATION, FORM LOCATION, ATTRIBUTE-TO-FIELD-MAPPING, FAILURE-FORMAT, HEADER, TAILER, ITEM-FORMAT>
608

FIG.8

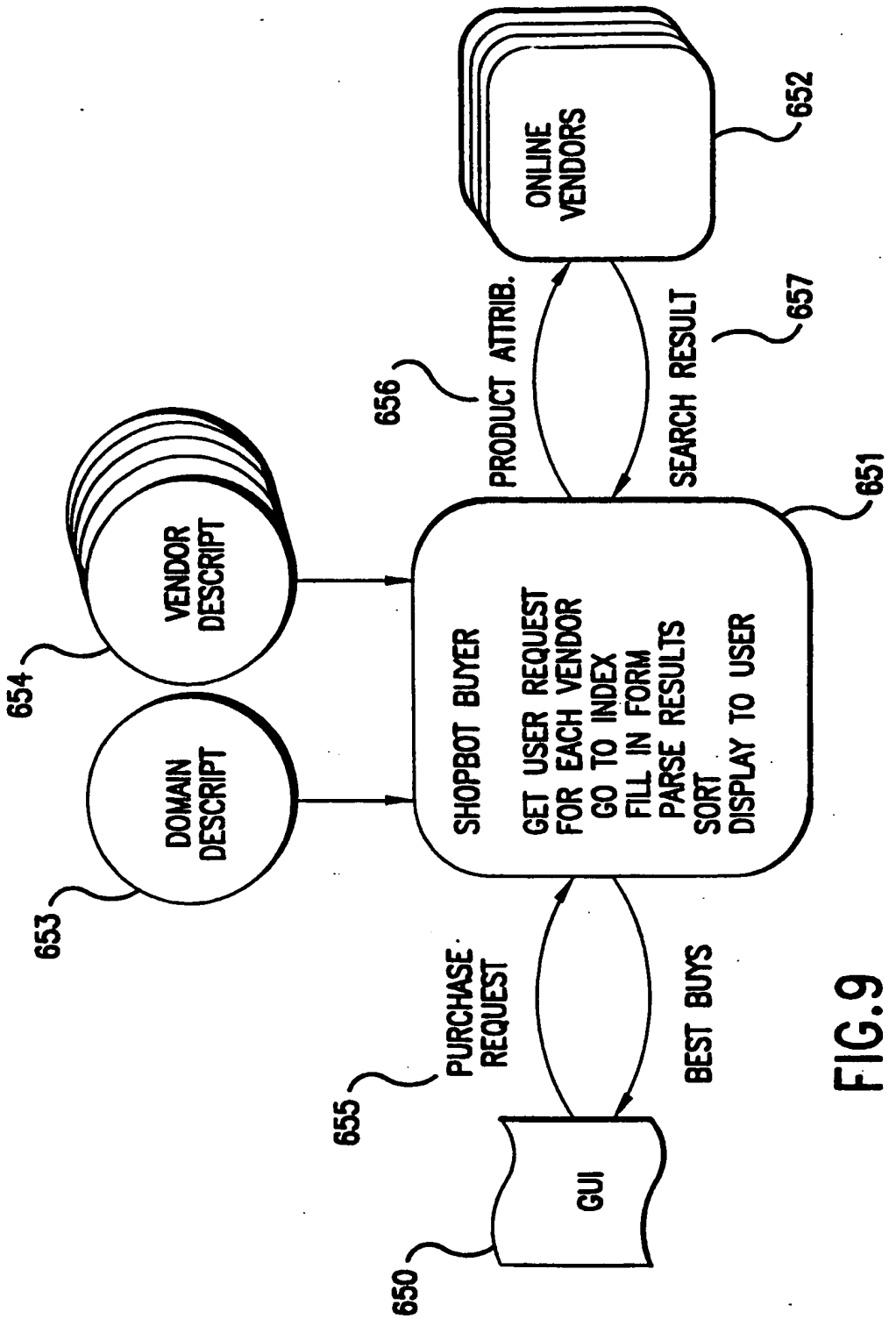

ONLINE VENDORS
652
PRODUCT ATTRIB.
656
SEARCH RESULT
657
VENDOR DESCRIPT
654
DOMAIN DESCRIPT
653
SHOPBOT BUYER
GET USER REQUEST
FOR EACH VENDOR
GO TO INDEX
FILL IN FORM
PARSE RESULTS
SORT
DISPLAY TO USER
651
PURCHASE REQUEST
655
BEST BUYS
GUI
650

FIG.9